Universidad Carlos III de Madrid

Escuela Politécnica Superior

PROYECTO FIN DE CARRERA

DISEÑO E IMPLEMENTACIÓN JAVA DEL EXTENSIBLE SERVICE REGISTRATION PROTOCOL (XSRP)

Autora:
Iria Quiroga García

Tutor:
Dr. Manuel Urueña Pascual

Leganés, diciembre 2009

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Motivación y Objetivos

En este proyecto se pretende llevar a cabo el diseño y la implementación del eXtensible Service Registration Protocol (XSRP), uno de los cuatro protocolos que conforman el eXtensible Service Discovery Framework (XSDF), una arquitectura para resolver los problemas de descubrimiento de servicio y de reparto de carga.

Existe un conjunto importante de tareas a desarrollar dentro de este Proyecto Fin de Carrera, que abarca disciplinas desde el análisis hasta la implementación pasando por el diseño de pruebas de conformidad.

Uno de los primeros objetivos será realiza un estudio extenso acerca de las diferentes alternativas para el descubrimiento de servicio, incluyendo sus ventajas e inconvenientes, y las razones por las cuales a día de hoy se encuentran en uso o no.

Otro de los objetivos importantes será el estudio en profundidad del conjunto de protocolos dentro de los cuales se enmarca el proyecto: el eXtensible Service Discovery Framework (XSDF). Para ello, se hará un especial hincapié tanto en la arquitectura que presenta como en cada uno de los diferentes protocolos que lo conforman y las operaciones y funcionalidades propias de cada uno.

Posteriomente será necesario, una vez se ha proporcionado una visión global del entorno en el que se enmarca este proyecto, una descripción en detalle el eXtensible Service Registration Protocol (XSRP), haciendo especial hincapié en como se produce el intercambio de mensajes, así como el formato de los mismos y las operaciones que ofrece a las aplicaciones finales.

Una vez definido el protocolo será necesario especificar el diseño que se ha seguido a la hora de realizar la implementación. Para ello, se hará uso de distintas herramientas, que irán desde diagramas de flujo que nos permitirán comprender la ejecución del protocolo, hasta diagramas que muestran, por un lado la estructura estática mediante diagramas de clase, como su comportamiento para lo que se hará uso de diagramas de comunicación, especialmente diagramas de secuencia, que permitan ilustrar el funcionamiento interno del desarrollo.

Dentro de todo desarrollo es necesario la realización de unas pruebas que aseguren su correcto funcionamiento, y nos permiten una cierta seguridad en su fiabilidad y su robustez, para ello se hará un estudio y diseño de las pruebas más importantes y necesarias para asegurar estas características en el ámbito de este proyecto.

1.2. Organización de la Memoria

La memoria está dividida en nueve capítulos. A continuación mostraremos un breve comentario sobre su contenido.

El capítulo dos presenta el estado del arte del área que nos concierne. En este caso comentaremos los diferentes protocolos de Descubrimiento de Servicio. Centrándonos en los siguientes:

- **Universal Plug and Play (UPnP)**: Por ser el protocolo de descubrimiento de servicio más utilizado y más extendido en el mercado.
- **Zeroconf - Bonjour**: Bonjour es la implementación de Apple de Zeroconf, varios estándares definidos por el IETF.
- **Service Location Protocol (SLP)**: Protocolo de descubrimiento de servicio en el que se basa XSLP, protocolo encargado del descubrimiento de servicios en XSDF.

El tercer capítulo trata sobre XSDF; esto es, el conjunto de protocolos que pretende integrar en una arquitectura común una solución para los problemas de descubrimiento de servicio y reparto de carga.

En el capítulo cuatro se describe el protocolo XSRP, tanto el formato de los mensajes como las operaciones que realiza.

En el capítulo cinco se detalla el diseño del cliente y servidor XSRP. En primer lugar se muestra la arquitectura de la que se parte, las librerías existentes, así como el diagrama de clases de uso y los diferentes casos de uso de la aplicación realizada.

En el capítulo seis se trata la implementación del eXtensible Service Registration Protocol (XSRP). Se describirá el conjunto de clases y métodos que se han implementado

En el capítulo siete se exponen los resultados de las pruebas realizadas a la implementación del protocolo, para comprobar el funcionamiento correcto del protocolo implementado, así como que se controlan tanto errores más frecuentes como cualquier otro tipo de error que puediera tener lugar.

En el capítulo ocho capítulo se comenta la gestión del proyecto. Esto incluye tanto la planificación temporal como el análisis de los costes.

Para finalizar, en el noveno y último capítulo se muestran las conclusiones que se pueden sacar del trabajo realizado así como los futuros trabajos que se pueden realizar a partir de este trabajo.

Capítulo 2

Estado del Arte

2.1. Introducción

En los últimos años, distintas empresas y desarrolladores de estándares han estado persiguiendo la configuración automática de dispositivos móviles para que pueden acceder automáticamente a los servicios de las redes a las que se conecta, actualmente conocido con el término *descubrimiento de servicio*. JINI[1], Universal Plug and Play[2] (UPnP), SALUTATION[3], Zeroconf[4] y Service Location Protocol (SLP)[6] son, quizá, los más conocidos. La elección de uno u otro dependerá de las necesidades y las soluciones requeridas soportadas por un usuario SOHO (Small office/Home office) o una gran corporación.

La movilidad y los entornos de computación ubicua han motivado la necesidad de los protocolos de descubrimiento de servicios mencionados. Con movilidad entendemos abandonar entornos configurados estáticamente y adentrarse en otras redes con infraestructuras desconocidas a priori. Además, como un dispositivo móvil no puede tener pre-configurada cada infraestructura a la que puede conectarse, no puede saber aprovecharse de ella o incluso tener la capacidad de interactuar con ella. Por ejemplo, un dispositivo móvil no puede usar una impresora cercana si no tiene el *driver* apropiado, o quizá una PDA experimentará un acceso lento a la Web por no tener acceso a la caché de un servidor Web que actúe como *proxy*.

A continuación haremos una introducción a los distintos protocolos de descubrimiento de servicio anteriormente mencionados, para tratar más adelante en profundidad los utilizados actualmente.

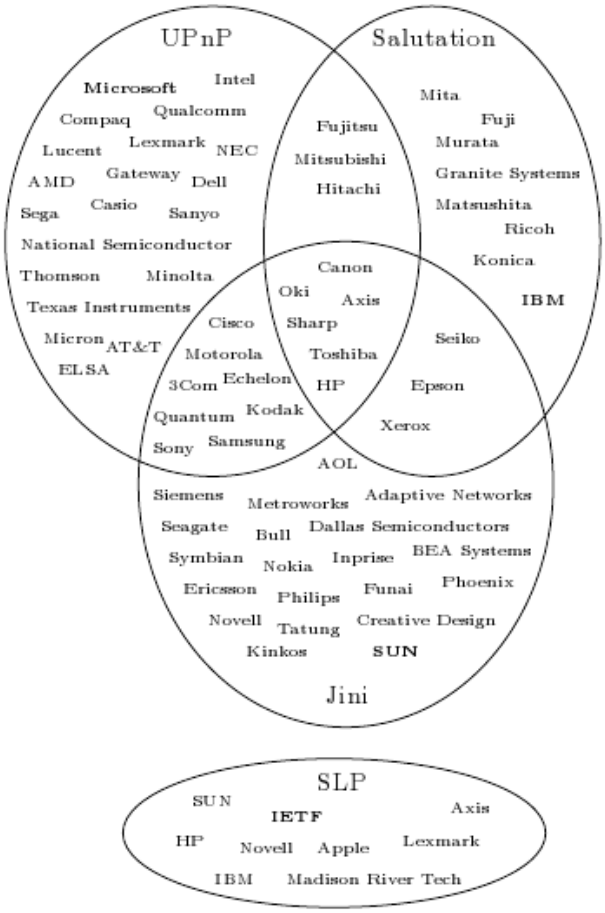


Figura 2.1: Empresas involucradas en el desarrollo de JINI, SALUTATION, UPnP y SLP

2.2. Jini

Sun Microsystems presentó Jini, basado en la tecnología Java, en 1998. La esencia de Jini es un trío de protocolos: descubrimiento, registro y búsqueda. Dos de esos protocolos (descubrimiento y registro) entran en funcionamiento cuando se conecta un dispositivo a la red; el descubrimiento ocurre cuando un servicio necesita la localización de otro en el que puede registrarse, mientras que el registro ocurre cuando un servicio localiza otro en el que quiere registrarse. La búsqueda ocurre cuando un cliente o usuario localiza e invoca un servicio descrito desde su interfaz tipo.

La tecnología Jini consiste en una infraestrutura y un modelo de programación que gestiona cómo los dispositivos se conectan para formar una comunidad. Jini usa Java Remote Method Invocation (RMI) para mover el código a través de la red.

Probablemente, la dependencia de Java sea una de las razones por las que Jini no alcanzó la popularidad deseada fuera de SUN Microsystems y sus socios directos.

2.3. SALUTATION

El Salutation Consortium desarrolló otro estándar, llamado SALUTATION, para descubrimiento de servicio, especialmente entre dispositivos y servicios de capacidades distintas. La arquitectura SALUTATION proporciona un método estándar para aplicaciones, servicios y dispositivos para describir y anunciar sus capacidades a otras aplicaciones, servicios y dispositivos. La arquitectura posibilita búsqueda y descubrimiento basado en capacidades particulares.

La arquitectura está compuesta de dos componentes: el *Salutation manager* y *Transport manager*. El *Salutation manager* es el núcleo de la arquitectura y es similar a la búsqueda de servicios en Jini o el punto de control de UPnP, que veremos más adelante.

El *Salutation manager* proporciona una interfaz de transporte independiente para las aplicaciones de cliente y servidor. Esta interfaz (SLM-API) incluye registro de servicios, descubrimiento de servicios y funciones de acceso a servicios. La interfaz entre el *Salutation manager* y *Transport manager* (llamado SLM-TI) obtiene independencia de protocolos de comunicación en la arquitectura SALUTATION. El *Transport manager* es una entidad, dependiente de la red de transporte empleada.

Este protocolo no tiene una importancia notable debido a que se dejó de trabajar en él y surgieron otros protocolos que cumplían su función y gozaron de mayor presencia en el mercado.

2.4. Universal Plug and Play (UPnP)

2.4.1. Introducción

Gracias a la tecnología Plug and Play (PnP) los Sistema Operativos, son mucho más fáciles de instalar, configurar y, en general, de agregar periféricos al PC. UPnP amplía esta simplicidad a toda la red permitiendo el descubrimiento y control de dispositivos y servicios de red, como por ejemplo, impresoras en red, routers de acceso, dispositivos wireless, PCs de todo tipo y equipos electrónicos del usuario.

UPnP es más que una simple extensión del modelo Plug and Play. Está diseñado para soportar la "configuración-cero", establecimiento de una red "invisible" y descubrimiento automático para una gran variedad de categorías de dispositivos de una amplia gama de proveedores.

Con UPnP un dispositivo puede unirse dinámicamente a una red, obtener una dirección IP, transmitir sus capacidades y aprender sobre la presencia y capacidades de otros dispositivos, todo automáticamente; permitiendo así, redes de "configuración-cero". Posteriormente, los dispositivos, pueden comunicarse directamente entre ellos.

El alcance de UPnP es suficientemente grande como para abarcar tanto escenarios actuales como otros nuevos incluyendo domótica, equipos de audio/vídeo, electrodomésticos, redes de automóviles, redes comunitarias en lugares públicos, etc.

UPnP usa estándares TCP/IP[7] y otros protocolos de Internet, permitiéndole encajar en las redes existentes. El uso de estos protocolos estandarizados permite a UPnP hacer de la interoperabilidad una característica inherente. Además, como UPnP es una arquitectura distribuida y abierta de red, definida por protocolos, es independiente del sistema operativo utilizado, lenguaje de programación o el medio físico. UPnP tampoco especifica los APIs de las aplicaciones que usará, permitiendo que los proveedores del sistema operativo creen los APIs que satisfagan las necesidades de sus clientes.

El UPnP Forum define el dispositivo y las descripciones de los servicios (originalmente llamado Device Control Protocols o DCPs) según una arquitectura común del dispositivo aportada por Microsoft. El UPnP Forum[2],

formado en 1999, lo componen más de 775 empresas, incluyendo líderes en la industria de la electrónica de consumo, informática, domótica, redes, movilidad, etc. Entre ellos algunos de los más populares: Siemens, Philips, IBM, Microsoft, Thomson, Motorola, Nokia, Intel, Honeywell y Ericsson.

El sitio Web del UPnP Forum *http://upnp.org* es el repositorio central para todas las especificaciones que ha sido desarrolladas y estandarizadas por el Forum. Incluyendo el documento de la arquitectura, los patrones para los dispositivos y las descripciones de servicios, y las pautas para los dispositivos y diseño de descripciones de servicios. También distribuye la información sobre las actividades y el progreso del Forum.

2.4.2. Componentes de la Red UPnP

Los componentes básicos de la red UPnP son los dispositivos, servicios y puntos de contol. Esta sección los describe con mayor detalle.

Dispositivos

Un dispositivo UPnP es un contenedor de servicios y otros dispositivos jerarquizados. Por ejemplo, un dispositivo VCR puede consistir en un servicio de grabación de cinta y un servicio de reloj. Un dispositivo combo TV/VCR consistiría no sólo en servicios independientes, sino también en un dispositivo jerarquizado.

Diversas categorías de los dispositivos UPnP están asociados con diferentes grupos de servicios y dispositivos integrados. Por ejemplo, los servicios incluidos en un VCR serán diferentes de los incluidos en una impresora. Consecuentemente, diferentes grupos de trabajo del UPnP Forum estandarizan el sistema de servicios que un tipo particular de dispositivos proporcionará. Toda esta información está capturada en un documento XML[8] de descripción de dispositivos que el dispositivo tiene que recibir. Además del sistema de servicios, la descripción del dispositivo también enumera las propiedades (tales como iconos y nombre del dispositivo) asociadas al dispositivo.

Servicios

La unidad de control más pequeña en una red UPnP es un servicio. Un servicio expone acciones y modela su estado con variables de estado. Por ejemplo, un servicio de reloj podría estar modelado como si tuviese una variable de estado "current_time", que define el estado del reloj, y dos acciones, "set_time" y "get_time", que permite controlar el servicio. De manera similar a la descripción del dispositivo, esta información forma parte de una descripción de servicio XML estadarizada por el UPnP Forum. Incluirá un puntero

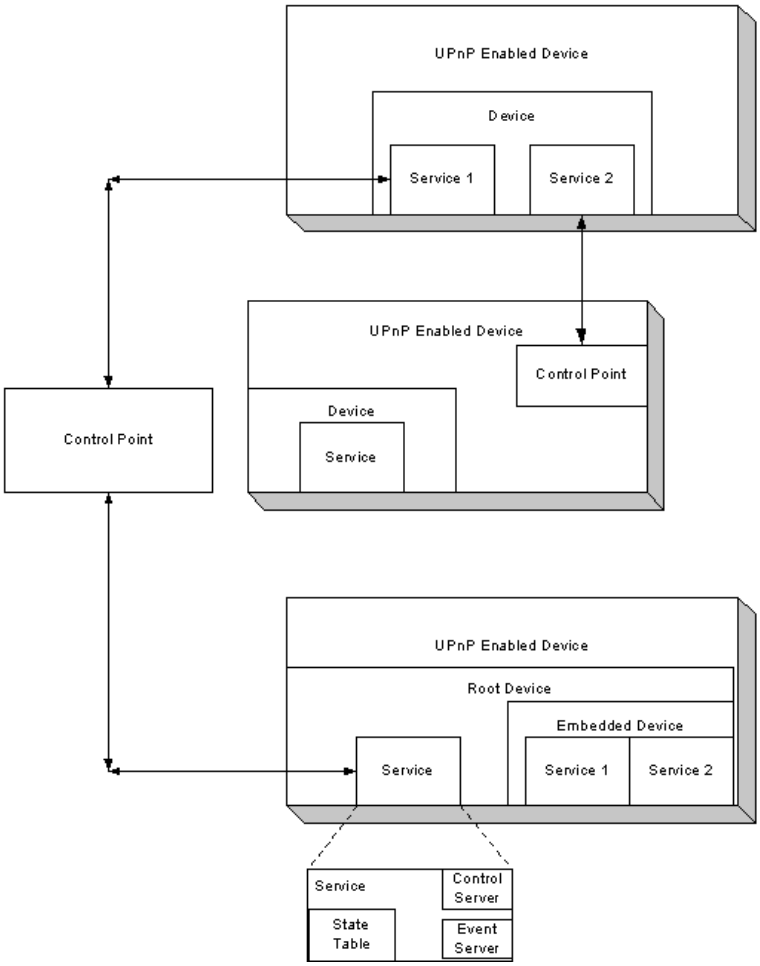


Figura 2.2: Puntos de Control, Dispositivos y Servicios

(URL) al documento de descripción del dispositivo. Los dispositivos pueden contener múltiples servicios.

Un servicio de un dispositivo UPnP consiste en una tabla de estado, un servidor de control y un servidor de eventos. La tabla de estado modela el estado del servicio con variables de estado y las actualiza cuando el estado cambia. El servidor de control recibe peticiones de acciones (como "set_time"), las ejecuta, actualiza la tabla de estado y devuelve una respuesta. El servidor de eventos genera eventos a los subscriptores interesados siempre que el estado del servicio cambia. Por ejemplo, el servicio de alarma anti-incendios enviaría un evento al subscriptor interesado cuando el estado cambia a "sonando".

Puntos de Control

Un punto de control en una red UPnP es un controlador capaz de descubrir y controlar otros dispositivos. Después del descubrimiento, un punto de control podría:

- Recuperar la descripción del dispositivo y conseguir una lista de servicios asociados.
- Recuperar las descripciones del servicio para los servicios interesados.
- Invocar las acciones para controlar el servicio.
- Suscribirse a la fuente del eventos del servicio. De manera que si el estado del servicio cambia en cualquier momento, el servidor de eventos enviará un evento al punto de control.

2.4.3. Visión General del Protocolo UPnP

UPnP usa protocolos estándares, tales como TCP/IP, HTML[9] y XML. Sin embargo, en los dispositivos de la red podrían usar otras tecnologías por otras razones, incluyendo coste, requisitos de la tecnología, o compatibilidad hacia atrás. Por ejemplo tecnologías del establecimiento de una red como HAVi, CeBus, LonWorks, EIB o X10. También pueden participar en la red de UPnP utilizando bridge o proxy. Una red UPnP conteniendo dispositivos bridged puede parecerse a la figura siguiente.

Algunos de los protocolos usados para la implementación de UPnP se resumen en el resto de esta sección:

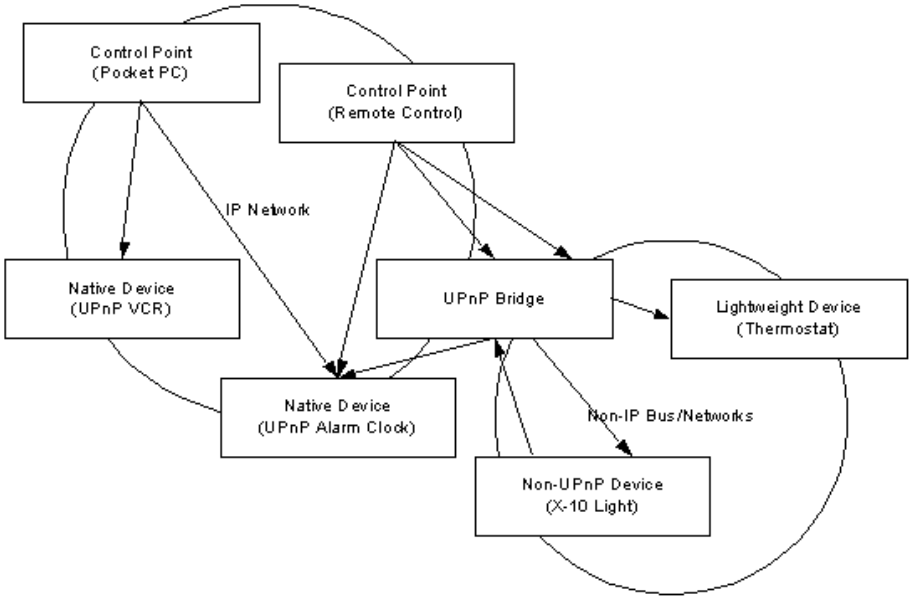


Figura 2.3: Una Red UPnP con Bridged

Protocolos Específicos UPnP

Los vendedores de UPnP, los comités de trabajo del UPnP Forum y el documento de la Arquitectura de Dispositivos de UPnP, definen las capas más altas de los protocolos usados para su implementación. De acuerdo con la arquitectura del dispositivo, los comités de trabajo definen las especificaciones refentes a los tipos de dispositivo tales como VCRs, sistemas de HVAC, lavavajillas, y otras aplicaciones. Posteriormente los vendedores de dispositivos UPnP añaden los datos específicos para sus dispositivos tales como el nombre del modelo, URL, etc.

TCP/IP

La pila de protocolos de red TCP/IP sirve como base sobre la que se asienta el resto de protocolos UPnP. De esta forma, UPnP aprovecha la capacidad de los protocolos TCP/IP de atravesar diversos medios físicos y asegura interoperabilidad entre múltiples vendedores.

Los dispositivos UPnP pueden utilizar muchos de los protocolos incluidos en la pila TCP/IP, como por ejemplo TCP, UDP, IGMP, IP, y ARP así como servicios TCP/IP tales como DHCP[10] y DNS[11]. Como estos protocolos y servicios se utilizan como base para que funcione UPnP, primero se aclarará como se utilizan los protocolos TCP/IP en esta sección y cómo trabaja UPnP en posteriores secciones.

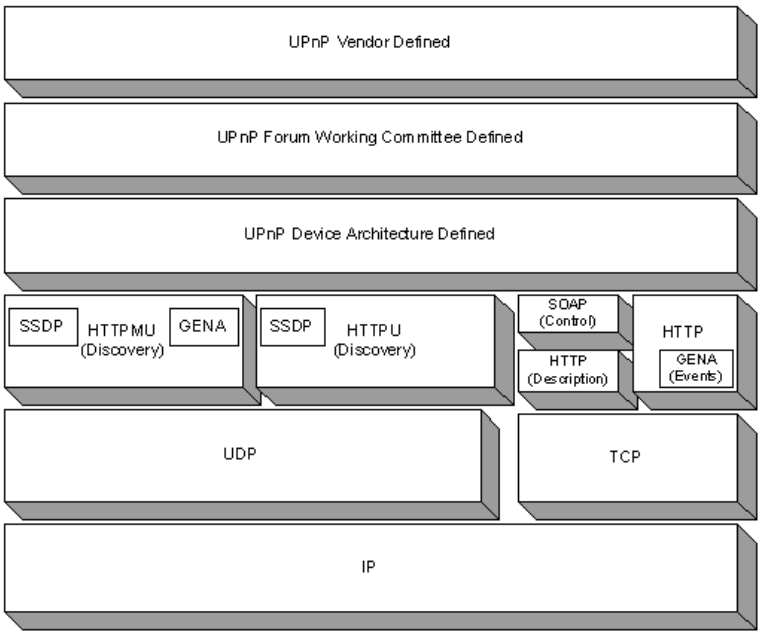


Figura 2.4: Torre de Protocolos UPnP

HTTP, HTTPU, HTTPMU

TCP/IP proporciona la base de la torre de protocolos para proporcionar la conectividad entre los dispositivos UPnP. HTTP[12], que es enormemente responsable del éxito de Internet, es también la pieza base de UPnP. Todos los protocolos de UPnP se basan en HTTP o sus variantes.

HTTPU (y HTTPMU) son variantes de HTTP definidos para la entrega de mensajes sobre UDP/IP en vez de TCP/IP. Estos protocolos son utilizados por el protocolo SSDP[13], descrito después. El formato básico de mensajes usado por estos protocolos es similar a HTTP y se utilizan para la comunicación multicast o cuando la entrega de mensajes no requiere el overhead asociado con fiabilidad.

SSDP

El Simple Service Discovery Protocol (SSDP), como su propio nombre indica, define cómo descubrir un servicio de red. SSDP esta construido sobre HTTPU y HTTPMU, y define métodos tanto para que un Punto de Control localice recursos de interés en la red, como para que los dispositivos anuncien su disponibilidad en la red. Mediante el uso de las peticiones de la búsqueda y de los avisos de la presencia, SSDP elimina los gastos indirectos que serían

necesarios si se utilizase solamente uno de estos mecanismos. Consecuentemente, cada Punto de Control de la red tiene información completa sobre el estado de la red, a la vez que mantiene un tráfico de red reducido.

Tanto los Puntos de Control como los Dispositivos UPnP usan SSDP. Un Punto de Control UPnP, durante el arranque, puede enviar un SSDP Search Request (sobre HTTPMU) para descubrir dispositivos y servicios que están disponibles en la red. El Punto de Control puede refinar la búsqueda para encontrar sólo un tipo particular de dispositivos (tales como un VCR), un sevicio concreto (como dispositivos con servicio de reloj), o incluso un dispositivo concreto.

Los dispositivos UPnP escuchan por el puerto multicast. Cuando se recibe un Search Request, el dispositivo examina los criterios de la búsqueda para determinar si hay coincidencia. Si se encuentra una coincidencia, se envía un Response unicast SSDP (sobre HTTPU) al punto de control.

De manera similar, un dispositivo conectado a la red, enviará múltiples mensajes SSDP Presence anunciando los servicios que soporta, que se almacenarán en la caché de los Puntos de Control para evitar consultas SSDP innecesarias.

Tanto los mensajes Presence como los mensajes unicast Response de los dispositivos contienen un puntero a la localización del documento de descripción del dispositivo que tiene información sobre el conjunto de propiedades y servicios soportados por el dispositivo.

Además de las capacidades del descubrimiento, SSDP también proporciona una manera para que un dispositivo y servicio asociado deje satisfactoriamente la red (Bye-Bye Notification) e incluye caché con temporizadores para limpiar información antigua.

GENA

La Generic Event Notification Architecture[14] (GENA), ha sido definida para poder enviar y de recibir notificaciones usando HTTP sobre TCP/IP y UDP multicast. GENA también define los conceptos de suscriptores y editores de notificaciones para permitir eventos.

Los formatos de GENA se utilizan en UPnP para crear los avisos Presence que se enviarán usando Simple Service Discovery Protocol (SSDP) y para proporcionar la capacidad de notificar cambios en el estado del servicio mediante eventos UPnP. Un Punto de Control interesado en la recepción de

notificaciones de eventos se suscribirá a una fuente de eventos enviando un Request que incluya el servicio de interés, una localización para enviar los eventos y una subscripción de tiempo para la notificación de eventos.

La suscripción debe renovarse periódicamente para continuar recibiendo notificaciones, aunque también puede cancelarse explícitamente usando GENA.

SOAP

El Simple Object Access Protocol (SOAP)[15], define el uso de XML y HTTP para la ejecución de llamadas remotas. SOAP se está convirtiendo en el estándar para la comunicación basada en RPC sobre Internet ya que permite el uso de la infraestructura existente en Internet como por ejemplo con firewalls y proxies. SOAP puede hacer uso de Secure Sockets Layer (SSL), para la seguridad y usa HTTP para la gestión de conexiones, de este modo, realizar comunicaciones distribuidas en Internet es tan fácil como el acceso a páginas web.

De manera similar a la gestión remota de llamadas, UPnP usa SOAP para la entrega de mensajes de control a los dispositivos y devuelve resultados de error a los puntos de control.

Cada UPnP Control Request es un mensaje SOAP que contiene la acción a invocar junto con un conjunto de parámetros. La respuesta es otro mensaje SOAP que contiene el estado del servicio y devuelve un valor y alguno de los parámetros.

XML

El Extensible Markup Language (XML), definido por el W3C (World Wide Web Consortium, consorcio internacional que produce estándares para la Web), es el formato universal para incluir datos estructurados en la Web. Por otra parte, XML es una manera de poner casi cualquier clase de datos estructurados en un archivo de texto.

XML se parece a HTML en que utiliza etiquetas y atributos. Realmente es bastante diferente ya que el significado de estas etiquetas y atributos no están definidas globalmente, sino se interpreta dentro del contexto de su uso. Estas características de XML lo hacen una buena elección para el desarrollo de esquemas para diferentes tipos de documentos.

XML es una parte fundamental de UPnP, y se usa en las descripciones de dispositivos y servicios, mensajes de control y sucesos.

2.4.4. Cómo Trabaja UPnP

Direccionamiento

El aspecto fundamental para el establecimiento de una red UPnP es el conjunto de protocolos TCP/IP y la clave de este conjunto es el direccionamiento. Cada dispositivo tiene que tener un cliente Dynamic Host Configuration Protocol (DHCP) y buscar un servidor DHCP cuando el dispositivo se conecta a la red por primera vez. Si un servidor DHCP está disponible, el dispositivo tiene que usar la dirección IP asignada por él. Si un servidor DHCP no está disponible, el dispositivo tiene que usar Auto-IP[24] para conseguir una dirección. Auto IP define cómo un dispositivo elige inteligentemente una dirección IP de un conjunto de direcciones privadas reservadas y puede moverse fácilmente entre redes gestionadas y sin gestionar.

Un dispositivo puede implementar protocolos de alto nivel, diferentes a UPnP, que usa nombres conocidos de dispositivos. En estos casos, es necesario permitir la resolución de nombres de host (dispositivos) a direcciones IP. El Domain Name Service (DNS) se usa normalmente para esto. Un dispositivo que requiere o usa esta funcionalidad puede incluir un cliente DNS y debe soportar registros dinámicos DNS para mapear su nombre a su de dirección actual.

Descubrimiento

Una vez que los dispositivos se unen a la red y comienza la fase de descubrimiento. Cuando un dispositivo se agrega a la red, SSDP permite que ese dispositivo anuncie sus servicios a los puntos de control de la red. Cuando un punto de control se agrega a la red, SSDP le permite buscar dispositivos de interés en la red.

El intercambio fundamental en ambos casos es un mensaje de descubrimiento que contiene algunas especificaciones esenciales sobre el dispositivo o uno de sus servicios, por ejemplo su tipo, identificador, y un puntero al documento XML de descripción del dispositivo.

Descripción

El paso siguiente en el establecimiento de una red UPnP es la descripción. Después de que un punto de control haya descubierto un dispositivo, todavía sabe muy poco sobre él. Para que aprenda más sobre éste y sus capacidades, el punto de control debe recuperar su descripción mediante la URL proporcionada por el dispositivo en el mensaje de descubrimiento.

Los dispositivos pueden contener otros dispositivos y servicios lógicos. La descripción de UPnP para un dispositivo se expresa en XML e incluye, la información del fabricante incluyendo el nombre modelo y el número, número de serie, nombre del fabricante, URLs de los sitios Web del vendedor, y así sucesivamente. La descripción también incluye una lista de cualquier dispositivo o servicio integrado, así como URLs para el control, eventos y la presentación.

Control

Después de que un punto de control haya recibido la descripción del dispositivo, tiene lo esencial para su control. Para aprender más sobre sus servicios, un punto de control debe obtener una descripción UPnP detallada para cada uno de ellos. La descripción para un servicio también se expresa en XML e incluye una lista de los comandos o acciones que ofrece el servicio, con los parámetros o argumentos para cada acción. La descripción para un servicio también incluye una lista de variables. Estas variables modelan el estado del servicio en tiempo de ejecución, y se describen en términos de su tipo de datos, y características del evento.

Para controlar un dispositivo, un punto de control envía una Action Request al servicio del dispositivo. Para hacer esto, envía un mensaje de control adecuado a la URL de control del servicio (proporcionado en la descripción del dispositivo). Los mensajes de control también se expresan en XML usando SOAP.

En respuesta al mensaje de control, el servicio devuelve valores de la acción o códigos de error específicos.

Eventos

Una descripción de UPnP para un servicio incluye la lista de acciones que el servicio ofrece y una lista de las variables que modelan el estado del servicio en tiempo de ejecución. El servicio publica actualizaciones cuando estas variables cambian, y un punto de control puede suscribirse para recibir esta información.

El servicio publica actualizaciones enviando mensajes de evento. Los mensajes del evento contienen los nombres de alguna de las variables de estado y del valor actual de dichas variables. Estos mensajes también se expresan en XML usando GENA.

Cuando un punto de control se subscribe por primera vez, se envía un mensaje de evento especial. Este mensaje contiene los nombres y valores de todas las variables de eventos y permiten que el subscriptor inicialice su modelo del estado de servicio.

Para soportar múltiples puntos de acceso, los mensajes de eventos se envían a todos los subscriptores. Éstos reciben los mensajes para todas las variables de eventos y se envían sin importar porqué ha cambiado la variable de estado (en respuesta a una action request o debido a un cambio de estado).

Presentación

Si un dispositivo tiene una URL para la presentación, entonces el punto de control puede recuperar una página web de esta URL, cargar la página en un navegador, y dependiendo de las capacidades de la página, ésta puede permitir que un usuario controle el estado del dispositivo y/o la visión del estado del mismo, dependiendo de las capacidades específicas de la página de presentación.

2.5. Zeroconf y Bonjour

La tecnología Zeroconf, más conocida por la implementación de Apple, *Bonjour*, es una de las soluciones más destacadas para descubrimiento de servicio en redes de área local. Zeroconf usa DNS multicast para alcanzar el objetivo de eliminar las configuraciones en el descubrimiento de servicio.

Zeroconf o Zero Configuration Networking es un conjunto de técnicas del IETF que permiten crear de forma automática una red IP sin configuración o servidores especiales. También conocida como Automatic Private IP Addressing o APIPA, permite a los usuarios sin conocimientos técnicos conectar ordenadores, impresoras de red y otros elementos y hacerlos funcionar. Sin Zeroconf, un usuario con conocimientos técnicos debe configurar servidores especiales, como DHCP y DNS, o bien configurar cada ordenador de forma manual.

El objetivo de Zero Configuration Networking (Zeroconf) es resolver el siguiente problema: cuando múltiples dispositivos IP están físicamente conectados, un dispositivo debería poder usar el servicio ofrecido por otro sin necesidad de configurar los dispositivos manualmente. Por ejemplo, cuando un usuario conecta dos ordenadores directamente, debería poder llevar a cabo la tarea de transmisión de ficheros simplemente inicializando la aplicación apropiada en ambos extremos. Las aplicaciones se deben descubrir sin que el usuario tenga la necesidad de decirles cómo encontrarse.

Hoy en día, la tecnología Zeroconf es una de las soluciones más extendidas para descubrimiento de servicio en redes de área local. *Bonjour* es la implementación de Zeroconf de Apple, y es una parte integral del sistema operativo MAC OS. *Bonjour* también se instala en una gran número sistemas operativos Windows[21], gracias a la popularidad de iTunes (aplicación de reproducción de música de Apple), el cual instala *Bonjour* como parte del proceso de instalación. Para plataformas UNIX, existe una implementación madura de código abierto de Zeroconf llamada Avahi[17] que viene preinstalado en distribuciones como Debian[18] y Ubuntu[19]. En lo referente al hardware, prácticamente todas las impresoras vendidas hoy en día soportan Zeroconf. El número de aplicaciones que implementan Zeroconf crece rápidamente, evidentemente motivado por el crecimiento de tipos de servicio Zeroconf definidos en [22].

2.5.1. Zeroconf, mDNS, DNS-SD y Bonjour

Existe bastante confusión sobre lo que significa el término Zeroconf. El término proviene del IETF Zero Configuration Networking Group[23], el cual estaba encargado de desarrollar una escificación de requisitos para redes en ausencia de configuración y administración. El grupo de trabajo identificó tres requisitos de redes de configuración "zero":

1. Asignación de direcciones IP sin un servidor DHCP
2. Resolución de nombres sin un servidor DNS
3. Descubrimiento de servicio local sin ningún servidor "rendezvous"

Para el primer requisito, el grupo de trabajo produjo el estándar de autoasignación de direcciones locales (RFC 3927) [24] el cual ha sido implementado por la mayoría de sistemas operativos actuales. El grupo de trabajo nunca alcanzó un consenso para contemplar el segundo y tercer requisitos.

Mientas tanto, Apple presentó *Bonjour*. Bonjour es la implementación de Multicast DNS (mDNS)[25] y los protocolos DNS Service Discovery (DNS-SD)[22], los cuales son la propuesta de Apple para el segundo y tercer requisito de Zeroconf. Debido a la expansión de Bonjour, el término Zeroconf se convierte en sinónimo con la abstracción de que Bonjour implementa los protocolos mDNS y DNS-SD. Nuestro uso del término Zeroconf sigue este "espíritu".

La asignación propia de direcciones en la red local descrito en la RFC 3927 establece los fundamentos de Zeroconf basándose en que la red IP está presente en el nivel de enlace.

El segundo requisito de Zeroconf es satisfecho por mDNS. Un demonio mDNS es esencialmente un servidor DNS multicast. Usa el mismo tipo de registros y la misma estructura de paquetes. De hecho, una petición DNS de una aplicación no podría decir si la respuesta proviene de un servidor mDNS o un servidor DNS unicast convencional. Sin embargo, hay algunas diferencias importantes:

- mDNS es ejecutado por todos los *host* de una red local mientras que un sistema DNS convencional se ejecuta en un solo *host* servidor.
- Las peticiones (*Queries*) son enviadas por multicast a todos los hosts en la red local usando UDP y el puerto 5353 en vez del 53, el puerto habitual para DNS.
- Todos los nombres de los archivos mDNS tienen que acabar en ".local.". La resolución de tales nombres se enrutan a mDNS por el sistema operativo.

Un demonio mDNS proporciona la resolución de nombre de host local usando una consulta de tipo A. Por ejemplo:

```
Toms-Computer.local. A 160.139.243.99
```

DNS-SD, junto con DNS, satisface el tercer requisito de Zeroconf. DNS-SD define las convenciones de registros de nombres para PTR, SRV y TXT llevados por los demonios de mDNS. Los registros PTR son usados para enumerar las instancias de servicio de un tipo en particular. La instancia del servicio es *mapeada* con los nombres de host y números de puertos usando registros SRV. Los registros TXT acompañan a los registro SRV para proporcionar información sobre las instancias de los servicios. El siguiente ejemplo ilustra el concepto:

```
_daap.:tcp.local. PTR Tom's Music._daap._tcp.local.
_daap._tcp.local. PTR Joe's Music._daap._tcp.local.

Tom's Music._daap._tcp.local. SRV 0 0 3689 Toms-Computer.local.

Tom's Music._daap._tcp.local. TXT ''Version=196613'' ''Password=false''
''Media Kinds Shared=3''

Toms-Computer.local. A 160.39.243.99
```

Esto es una representación textual de varios registros DNS producidos por el programa de reproducción de música iTunes de Apple cuando la opción de compartir música está habilitada. Los registros PTR son usado para enumerar las dos instancias de servicio (la música de Tom y la música de Joe) que está disponible actualmente en la red local para el tipo de servicio "_daap._tcp". El nombre del host y el número de puerto para la instancia de un servicio específico (la música de Tom en este caso) lo proporciona un registro SRV. Un registro TXT con el mismo nombre que el registro SRV proporciona información adicional sobre la instancia del servicio. Finalmente, un registro A mapea el nombre de host para una dirección IP.

El demonio de mDNS se ejecuta en cada host en una red local almacenado colectivamente y administrando los registros PTR, SRV, TXT y A para el registro de servicios en la subred local. Las peticiones (queries) y respuestas se intercambian vía red local multicast.

2.6. Service Location Service (SLP)

El Service Location Protocol es un estandar del Internet Engineering Task Force (IETF), descentralizado, ligero y "extensible" de descubrimiento de servicio. Usa URLs de los servicio, que definen el tipo de servicio y direccionan un servicio particular. Por ejemplo "service:printer:lpr://hostname" es una URL para un servicio de "impresión" disponible en "hostname". Basado en la URL del servicio, los usuarios (o aplicaciones) pueden ver servicios disponibles en su dominio y seleccionar y usar el que se desee.

Figura 2.5: Esquema de la arquitectura SLP

Hay tres agentes en SLP: de usuario, de servicio y de directorio. Su arquitectura se puede observar en la figura 2.6

El objetivo del protocolo SLP es la localización dinámica de servicios, dentro de una red local, con un dominio administrativo común. Mediante este protocolo se intenta evitar que los servicios tengan que estar pre-configurados en cada máquina cliente, o que los usuarios tengan que recordar en qué máquina se encuentra cada servicio. Por el contrario, los usuarios tan sólo tienen que indicar el tipo y los atributos del servicio que desean emplear, y mediante SLP es posible encontrar los servicios activos en la red con dichas características. Una vez localizado el servicio, se puede acceder a él mediante cualquier otro mecanismo de comunicación, puesto que SLP se limita simplemente a encontrar los servicios para los clientes.

SLP puede emplearse tanto en subredes sin infraestructura, como en la red de una casa o de una pequeña oficina, o en redes de empresas con múltiples subredes, en las que es necesario desplegar cierta infraestructura para centralizar las búsquedas de servicios. Sin embargo SLP no es aplicable para localizar servicios en redes WAN como Internet o en redes con cientos de miles de clientes o decenas de miles de servicios.

En la arquitectura SLP existen tres tipos de entidades:

- **User Agent (UA)**. Un Agente de Usuario es una entidad que emplea SLP para obtener información sobre los servicios, normalmente su URL, en nombre de un usuario/cliente. El Agente de Usuario obtiene la información de los servicios de los Agentes de Servicio o de los Agentes de Directorio.
- **Service Agent (SA)**. Un Agente de Servicio es un proceso que trabaja en nombre de uno o más servicios para publicar la información de dichos servicios. Además, el Agente de Servicio es el encargado de registrar la información de sus servicios en el Agente de Directorio, si este se encuentra presente.
- **Directory Agent (DA)**. Un Agente de Directorio es una entidad que recoge la información de los Agentes de Servicio para ser un repositorio centralizado de información de servicios, y de esta forma optimizar las consultas de los Agentes de Usuario.

La operación básica del protocolo consiste en un cliente que quiere localizar un servicio dentro de la red. Para ello el servicio delega en un Agente de Servicio para publicar su información, mientras que el Cliente emplea a un Agente de Usuario para realizar la búsqueda. En redes de tamaño reducido,

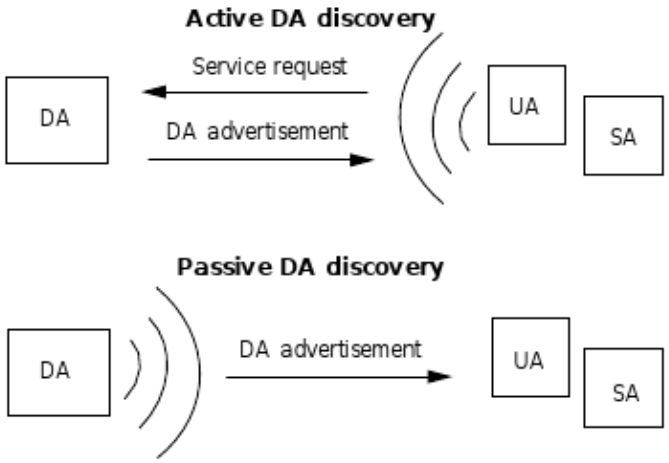


Figura 2.6: Métodos de descubrimiento de DA

el Agente de Usuario envía la consulta SLP a un grupo multicast conocido (o en *broadcast* si la red no soporta *multicast*). Los Agentes de Servicio que estén subscritos a ese grupo multicast responden en unicast, directamente al UA, con los servicios que se adecúen a la consulta formulada.

En redes de mayor tamaño es necesario limitar el alcance de las consultas ya que una gran cantidad de mensajes *multicast/broadcast* podría saturar la red. Para ello se introduce el concepto de Agente de Directorio, a modo de repositorio centralizado. Los Agentes de Usuario preguntan directamente al DA por la información de los servicios. De esta forma la comunicación es *unicast* y es posible emplear TCP en lugar de UDP, que se usa para las consultas *multicast*. Para que este sistema centralizado funcione los Agentes de Servicio deben registrar previamente la información de sus servicios en el DA y opcionalmente de-registrarlos si los servicios dejan de estar accesibles.

Para dividir redes de gran tamaño en unidades organizativas, tales como departamentos o grupos de trabajo, cada servicio puede asignarse a diferentes Grupos (*Scopes*). De esta forma es posible tener múltiples Agentes de Directorio en una red, donde cada uno se encarga de uno o varios Grupos, y los Agentes de Servicio solamente registran los servicios en los DAs encargados de sus Grupos. Las consultas de los Agentes de Usuario también están limitadas a determinados Grupos, de forma que se envían al DA de cada Grupo, y estos sólo devolverán los servicios pertenecientes a los Grupos especificados en la consulta.

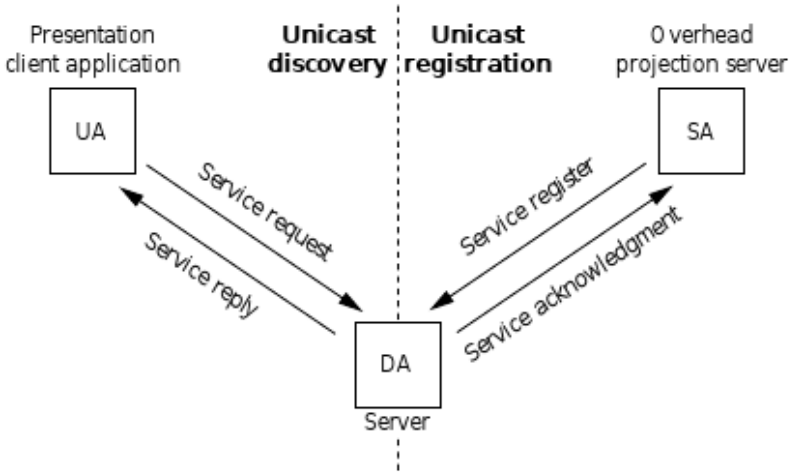


Figura 2.7: Registro de un servicio en el DA por un SA y peticion de un servicio por el UA en el DA

De los dos tipos de funcionamiento, el primer modo (consultas SLP en multicast) está limitado a redes LAN pequeñas, y no es necesario ningún tipo de configuración o de infraestructura. Por otro lado, al dividir los servicios de la red en múltiples Grupos y desplegar varios Agentes de Directorio, es posible aplicar SLP en redes que abarquen varias LAN y tengan un número elevado de usuarios.

El uso de un Agente de Directorio requiere que tanto los Agentes de Usuario como los Agentes de Servicio deben conocer previamente la situación del mismo. Para ello se han definido varios mecanismos: configuración manual, mediante DHCP (descrito mas adelante), escuchando los anuncios que periódicamente envían los DAs, o realizando consultas a un grupo multicast de una forma parecida a la descrita anteriormente para la localización de servicios.

La figura 2.6 ilustra los dos métodos diferentes para el descubrimiento de DA: activo y pasivo. En el descubrimiento activo, UAs y DAs envían SLP multicast requests en la red. En el descubrimiento pasivo los DAs envían *multicast advertisements* a sus servicios y continúa haciendo esto periódicamente en el caso de que algún UA o SA haya fallado al recibir el advertisement inicial.

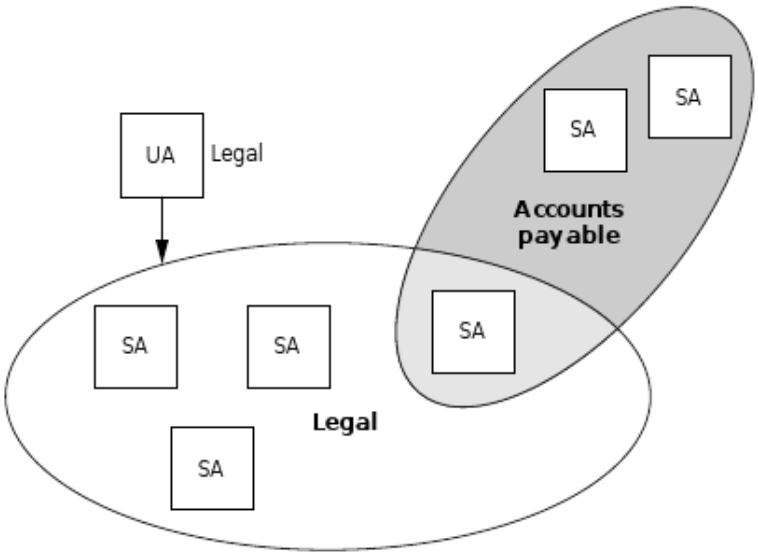


Figura 2.8: Scopes en SLP. Un UA sólo puede localizar servicios en el scope al que tiene acceso

Los UAs y SAs también pueden descubrir la localización de los DAs utilizando las opciones de DHCP para la localización de servicios (SLP DA Option (78)) [29]. Los servidores DHCP configurados por administradores de redes, pueden usar la opción 78 de distribuir las direcciones de DAs a los hosts que les preguntan. Los agentes SLP configurados de esta forma no necesitan usar descubrimiento multicast, ya que esto se utiliza solamente para el descubrimiento de DAs y descubrimiento de servicios en ausencia de DAs.

Capítulo 3

eXtensible Service Discovery Framework (XSDF)

3.1. Introducción

En este capítulo se describen las características del eXtensible Service Discovery Framework (XSDF) haciendo en primer lugar una introducción a las característica y mecanismos de descubrimiento de servicio de SLP y de reparto de carga de Rserpool, que están íntimamente relacionados con XSDF. Además de esto se detallará el modelado de los servicios en XSDF, la mejora que supone la codificación con XBE32 con respecto a SLPv2 y XML, la organización en grupos administrativos para la gestión de los usuarios así como la descripción de cada uno de los protocolos que forman parte de XSDF.

Para la realización de este apartado se ha utilizado [40] como principal fuente de información.

3.1.1. Descubrimiento de Servicio: SLP

Tal como se estudió en el capítulo anterior, los múltiples beneficios del descubrimiento de servicios han fomentado el desarrollo de múltiples protocolos, como SLP, UPnP o Bonjour. Una de las principales diferencias entre SLP y otros protocolos reside en sus escenarios de aplicación. Mientras que la mayoría de los protocolos de descubrimiento de servicios están enfocados en pequeñas redes LAN sin administración, como una casa o una pequeña oficina, SLP ha sido diseñado cuidadosamente para poder escalar desde esas pequeñas LANs a grandes redes corporativas.

Esta escalabilidad se obtiene combinando diversos mecanismos. En redes pequeñas sin infraestructura estable, los Agentes de Usuario (UA) localizan

servicios enviando consultas multicast a los Agentes de Servicio (SA), que residen en cualquier ordenador que ofrezca un servicio. Sin embargo cuando el número de usuarios y servicios aumenta, el tráfico *multicast* puede sobrecargar la red. En ese caso, es recomendable desplegar una entidad opcional de la arquitectura, denominada Agente de Directorio (DA), que actúa como un repositorio centralizado de información de servicios, y donde los Agentes de Servicio publican sus servicios. Esto permite a los Agentes de Usuario enviar sus consultas directamente al Agente de Directorio empleando *unicast*. Por tanto, no es necesario que los Agentes SLP empleen consultas *multicast*, si no es para localizar inicialmente al Agente de Directorio (aunque también es posible emplear DHCP para configurar la localización del DA). Para poder escalar aún más, o cuando una organización está dividida en varios departamentos, es posible dividir a conjuntos de usuarios y servicios en varios Grupos[1] independientes, que pueden tener su propio Agente de Directorio.

Además, SLP consigue esta escalabilidad excepcional sin sacrificar su simplicidad. Sin embargo, a veces tanta simplicidad se convierte en un problema, y un buen ejemplo de ello es el modelado de los servicios en SLP. En SLP los servicios se localizan mediante su URL, y opcionalmente pueden tener una lista de pares atributo-valor para poder describirlos en más detalle. Además, la URL también se emplea como identificador del servicio. En [6], Guttman estudia los problemas derivados de este modelo de servicio excesivamente simple:

1. Un servicio está atado a una única localización, y por tanto no puede tener varias direcciones, ni cambiar de dirección IP.

2. No puede accederse a un único servicio mediante varios protocolos de aplicación (e.g. lpr, ipp).

3. Las URLs no permiten indicar qué protocolos de transporte pueden emplearse para acceder al servicio (e.g. SCTP).

Para solventar estos problemas, en [6] Guttman sugiere identificar a los servicios con una URN en lugar de una URL, que tan sólo contiene un UUID (Universally Unique Identificier) [26], mientras que la información sobre los protocolos de transporte y aplicación soportados, su localización y otra información adicional (nombre, descripción, etc) se envían como atributos asociados al servicio. Aunque esta modificación permite mantener el formato de los mensajes SLP, los Agentes de Usuario deben cambiar su comportamiento estándar cuando encuentran servicios de este tipo. Además, añadir tantos atributos (codificados en texto) a cada servicio puede suponer una gran sobrecarga.

[1] *Scopes*, en inglés

3.1.2. Reparto de carga en *clusters*: Rserpool

Dado que replicar un servicio en múltiples servidores se ha convertido en un mecanismo bastante común para conseguir escalabilidad y alta disponibilidad, el reparto de carga entre servidores desacoplados ha sido un tema muy popular, tanto en la literatura académica como en la comercial, y especialmente en el campo de los servidores web [27].

Los diseños iniciales con varios servidores web empleaban el DNS para balancear la carga entre ellos. Cuando los clientes pedían al servidor DNS autoritativo la dirección IP del servidor web, este devolvía cada vez la IP de un servidor diferente (Round-Robin). Sin embargo, la cache de los servidores DNS locales (o de los proxies web) producía un reparto de carga desigual entre servidores, especialmente en las situaciones de alta carga [28].

Como se considera que la técnica de DNS Round-Robin permite realizar un reparto de carga bastante tosco, muchas organizaciones han desplegado Conmutadores L4/L7 ("Balanceadores de Carga") como primera capa en sus clusters de servidores Web para conseguir una mejor distribución de carga. Estos elementos se comportan como un NAT, ocultando los servidores finales, e implementan diferentes algoritmos de selección para escoger al mejor servidor basándose en métricas tales como la capacidad del servidor, su carga actual, el tiempo de respuesta o el número de conexiones de clientes. Además, algunos dispositivos capturan la cabecera HTTP de las peticiones y son capaces de realizar esta selección basándose en la URL de la petición u otra información sobre el servicio solicitado.

Un mecanismo alternativo para repartir la carga, es la selección de servidor en el cliente, que a diferencia de los anteriores, se ajusta perfectamente con la arquitectura extremo-a-extremo de Internet. Quizá el grupo de trabajo Rserpool del IETF es el mejor ejemplo de este tipo de técnicas. Rserpool define una arquitectura [30], muy similar a la de SLP, donde a un conjunto de servidores que proporcionan el mismo servicio se le denomina Pool. Para poder acceder a alguno de los servidores del cluster, el Usuario del Pool (PU) pide al gestor del cluster -denominado Servidor ENRP- todos los Elementos del Pool (PE) disponibles, y el PU elige al mejor de ellos.

Dado que la política de selección depende en gran medida del tipo de servicio solicitado, Rserpool es extensible y permite diferentes mecanismos y métricas de reparto de carga. Además, dado que el cliente conoce varios servidores que ofrecen el mismo servicio, en caso de fallo puede seleccionar un servidor alternativo, empleando el protocolo ASAP [31]. Dado que Rserpool está orientado a clusters con alta disponibilidad, es posible emplear varios servidores ENRP coordinados mediante el protocolo del mismo nombre [32].

Rserpool emplea obligatoriamente SCTP como protocolo de transporte. Sin embargo, aún no se ha estudiado el efecto que puede tener sobre servicios interactivos el retardo introducido por establecimiento de la conexión SCTP para consultar el servidor ENRP. Por ejemplo, en la navegación Web la resolución de nombres DNS es mucho más rápida y, aún así, puede afectar en gran medida a la percepción del usuario sobre el rendimiento del servicio.

3.1.3. Reparto de Carga Global

La replicación de servicios puede ir un paso más allá y no sólo emplear múltiples servidores, sino también múltiples sedes repartidas por todo el mundo. Normalmente en este escenario se necesitan varias técnicas de reparto de carga: primero una política de reparto de carga global para seleccionar la sede más cercana y luego una política de reparto de carga local para elegir al servidor menos cargado de dicha sede.

Una forma bastante habitual para implementar el reparto de carga global consiste en emplear un servidor DNS autoritativo modificado. Cuando un cliente le envía una consulta DNS sobre la página web de la organización, éste devuelve la dirección IP del balanceador de carga de la sede más cercana al usuario. Después, el balanceador de carga redirige la conexión del cliente al mejor servidor dentro de la sede. Sin embargo, dado que la conexión del cliente -y por tanto la petición HTTP- sólo se establece con el balanceador de carga de la sede escogida, la selección a nivel global no puede basarse en el contenido de la petición (e.g. la URL), sino meramente en la consulta DNS inicial, a no ser que se emplee una conexión triangular entre el cliente, la primera sede (que realiza la selección definitiva de sede basada en el contenido de la petición HTTP), y el balanceador de carga de la sede definitiva.

Otro problema de esta técnica consiste en encontrar la sede más "cercana" al cliente desde la perspectiva del proveedor de servicios. Algunas heurísticas incluyen: la consulta de tablas de rutas BGP, el envió de pings desde todas las sedes al cliente o la recopilación de información estadística sobre conexiones anteriores. Ninguna de estas técnicas es sencilla y, lo que es peor, ni siquiera ofrecen garantías de proximidad efectiva, sobre todo si tenemos en cuenta que la dirección IP que se emplea como índice para el reparto de carga global no suele ser directamente la del cliente final sino la de su servidor de DNS local que, a pesar de lo que pueda parecer, en los ISPs suele estar bastante alejado del cliente final.

3.2. eXtensible Service Discovery Framework (XSDF)

El *eXtensible Service Discovery Framework* (XSDF) ha sido diseñado para intentar resolver los dos problemas, tanto el descubrimiento de servicios,

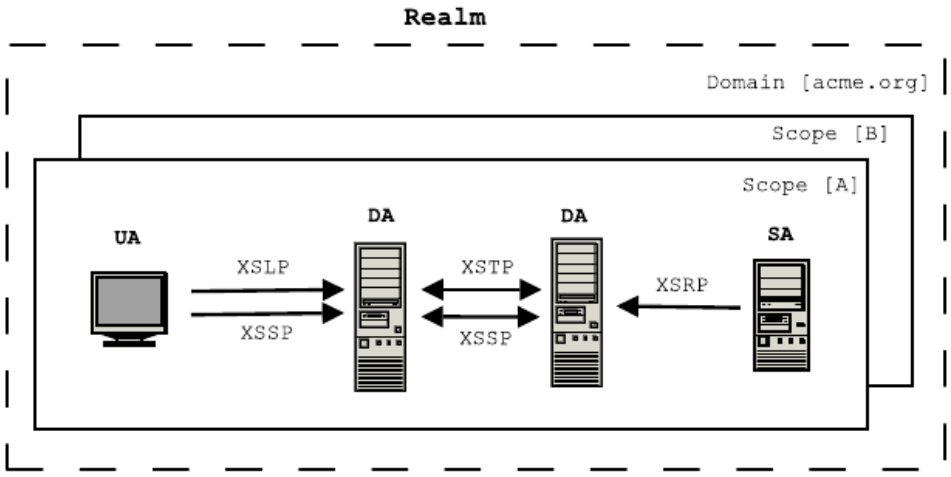


Figura 3.1: Arquitectura XSDF y sus protocolos

como el reparto del carga, puesto que integra en una arquitectura común ambas soluciones. Como otros entornos de descubrimiento de servicios, y tal como se muestra en la Figura 3.1, XSDF está compuesto por varios agentes y por los protocolos que se emplean para comunicarse entre sí.

Al igual que en SLP, XSDF tan sólo se ocupa del descubrimiento de servicios (y adicionalmente de la selección del mejor servicio). Una vez que un UA ha escogido un servicio, puede emplear cualquier otro protocolo para comunicarse con él. Por tanto, XSDF tan sólo permite descubrir la información de localización y los protocolos que pueden emplearse posteriormente.

En particular, XSDF hereda la misma arquitectura y nomenclatura de SLP, aunque hay varias diferencias entre ambos. Quizá la más visible sea que XSDF define varios protocolos muy simples, en lugar de uno sólo más complejo, como en el caso de SLP. Cada uno de estos protocolos cliente-servidor se encarga de una de las partes que forman el proceso de descubrimiento de servicios:

- ***eXtensible Service Location Protocol* (XSLP)** [34]: Los Agentes de Usuario (UA) emplean este protocolo para obtener información de los Agentes de Servicio (SA) o de los Agentes de Directorio (DA). XSLP es el protocolo principal de XSDF y permite tanto consultas *unicast* al DA, como consultas *multicast* a todos los SAs de la red, cuando no se han desplegado DAs en la red.

- ***eXtensible Service Registration Protocol*(XSRP)** [35]: Cuando un grupo de trabajo está gestionado por uno o más Agentes de Directorio, los Agentes de Servicio deben emplear XSRP para registrar la información de sus servicios en el DA. Por tanto, XSRP permite agregar toda la información de servicios en un directorio central, lo que posibilita las búsquedas XSLP *unicast*.

- ***eXtensible Service Subscription Protocol* (XSSP)** [36]: Este protocolo permite a cualquier Agente XSDF suscribirse a los cambios en la información de servicios, mediante el registro de un canal de notificaciones. Todos los cambios que afecten a la información de servicios asociada se envían a dicho canal, que puede ser un Agente individual o una dirección multicast para distribuir los eventos a varios Agentes. Los Agentes de Directorio con grupos comunes emplean este protocolo para suscribirse a las operaciones XSRP que afectan a dichos grupos.

- ***eXtensible Service Transfer Protocol* (XSTP)** [37]: Para conseguir un servicio de descubrimiento de servicios escalable y de alta disponibilidad, cada grupo de trabajo puede estar gestionado por varios Agentes de Directorio. Estos DAs emplean el protocolo XSTP cuando se inicializan para obtener de otro DA toda la información de servicios disponible. Posteriormente también se emplea para sincronizar el repositorio de servicios. Para ello basta con reenviar todas las operaciones XSRP recibidas, que afecten al grupo de trabajo común, al resto de DAs, que se subscriben mediante XSSP y emplean un canal de notificaciones XSTP para encapsular dichos mensajes XSRP.

Aunque XSDF puede parecer más complejo que SLP o Rserpool, que sólo emplean uno y dos protocolos respectivamente, consideramos que esta separación en cuatro protocolos redunda en una arquitectura más simple. Por ejemplo, tanto en SLP como en Rserpool, las consultas de los UAs/PUs y las operaciones de registro de los SAs/PEs están definidas en el mismo protocolo (SLP/ASAP), aunque cada tipo de operación tiene unas funcionalidades, requisitos de seguridad y agentes origen/destino muy diferentes entre sí. Al separar estas operaciones disjuntas en dos protocolos, es mucho más sencillo gestionar la infraestructura de descubrimiento de servicios. Por ejemplo, bastaría con filtrar todos los puertos XSDF en el firewall de la organización, excepto la consultas XSLP dirigidas a sus DA públicos, para implementar una política de seguridad perimetral sencilla pero muy eficaz.

Además, la utilización de varios protocolos no implica que los Agentes tengan que ser más complejos, puesto que cada tipo de Agente tiene que implementar solamente la parte cliente o servidora adecuada y además, dependiendo del escenario de aplicación, algunos de los protocolos XSDF (i.e.

XSRP, XSSP, XSTP) pueden ser innecesarios. Por ejemplo los Agentes de Usuario podrían limitarse a soportar solamente la parte cliente de XSLP, mientras que los Agentes de Directorio tienen que implementar XSLP y la parte cliente de XSRP. Por tanto, los protocolos más complejos: XSSP, XSTP y el servidor de XSRP tan sólo son necesarios cuando se emplean Agentes de Directorio, que además serán servidores especializados, a diferencia de los UAs y SAs, que pueden ejecutarse en cualquier dispositivo y por tanto deben simplificarse al máximo.

3.3. Modelado de los Servicios

En SLP un servicio se define con una URL, y una lista opcional de pares atributo-valor textuales (figura 3.4). Por el contrario, en Rserpool la información de los servicios tiene un formato binario, ya que está compuesta por una o más direcciones de transporte, esto es, un puerto y una o más direcciones IP por cada protocolo de transporte soportado, además de ciertos datos para la política de selección.

El modelo de servicio en XSDF mezcla ambas aproximaciones en una única representación extensible de servicio (figura 3.2), que está dividida en cinco partes bien diferenciadas:

- **Identificador de Servicio**: Cada instancia de un servicio está identificada por un UUID de 128 bits, que debe ser único dentro del Dominio.

- **Estado del Servicio**: Contiene datos volátiles que no deberían almacenarse en la cache de los Agentes de Usuario durante demasiado tiempo (e.g. la carga de trabajo actual del servidor). Además, este apartado contiene campos con la versión del resto de la información, de forma que sea posible detectar si ha cambiado simplemente consultando esta parte de los datos.

- **Información principal del Servicio**: Dentro de estos datos se incluye el Tipo del Servicio, su Política de Selección y otra información importante dependiente del servicio, como por ejemplo si una impresora soporta impresión a doble cara o a color.

- **Información de Localización del Servicio**: Para poder comunicarse con un servicio de red, los clientes deben saber dónde pueden localizarle, y qué protocolos pueden emplear para acceder al servicio. La información de localización incluye esos datos, tales como las direcciones IPv4/IPv6 del servidor, los puertos de nivel de transporte o qué protocolos de nivel de aplicación están soportados.

```
<service>
  <id>8e9d7823-d5ac-497c-91d0-fb07ea0c3fb2</id>

  <serviceState>
    <metaInfo>
      <stateTimestamp>f85444f4eb</stateTimestamp>
    </metaInfo>
    <selectState>
      <workload>0</workload>
    </selectState>
  </serviceState>

  <serviceMainInfo>
    <serviceType>
      <type>printer</type>
    </serviceType>
    <alias>Alice's printer</alias>
    <selectInfo>
      <policies>Least Used (0x0002)</policies>
      <weight>14</weight>
    </selectInfo>
    <printer:color>false</printer:color>
    <printer:duplex>true</printer:duplex>
  </serviceMainInfo>

  <serviceLocation>
    <inet>
      <ipv4Addrs>169.254.85.139</ipv4Addrs>
      <ipv6Addrs>fe80::202:b3ff:fe3c:da7a</ipv6Addrs>
    </inet>
    <protocol>
      <name>ipp</name>
      <transPorts>tcp/631, sctp/631</transPorts>
    </protocol>
    <protocol>
      <name>lpr</name>
      <transPorts>tcp/515, sctp/515</transPorts>
    </protocol>
  </serviceLocation>

  <serviceAddInfo>
    <model>Acme Laser Printer 2000</model>
    <modelURL>http://www.acme.org/printers/lp2000.html</modelURL>
  </serviceAddInfo>
</service>
```

Figura 3.2: Servicio XSDF de una impresora codificado con XML

```
01000284 35110014  8e9d7823 d5ac497c    ....5... ..x#..I|
91d0fb07 ea0c3fb2  01100020 0111000f    ......?. ... ....
331a000c 000000f8  5444f4eb 0311000c    3.......  TD......
32320008 00000000  0120006c 0121000f    22......  . .l.!..
2812000b 7072696e  74657200 28140013    (...prin  ter.(...
416c6963 65277320  7072696e 74657200    Alice's   printer.
03210014 31330008  00020000 32350008    .!..13..  ....25..
0000000d 10000018  20020009 636f6c6f    ........   ...colo
72000000 30020005  00000000 10000018    r...0...  ........
2002000a 6475706c  65780000 30020005     ...dupl  ex..0...
ff000000 01300054  01310020 32150008    .....0.T  .1. 2...
a9fe558b 35160014  fe800000 00000000    ..U.5...  ........
0202b3ff fe3cda7a  01320018 28610007    .....<.z  .2..(a..
69707000 321a000c  00060277 00840277    ipp.2...  ...w...w
01320018 28610007  6c707200 321a000c    .2..(a..  lpr.2...
00060203 00840203  0140008c 2867001b    ........  .@..(g..
41636d65 204c6173  65722050 72696e74    Acme Las  er Print
65722032 30303000  2868002c 68747470    er 2000.  (h.,http
3a2f2f77 77772e61  636d652e 6f69672f    ://www.a  cme.org/
7072696e 74657273  2f6c7032 3030302e    printers  /lp2000.
68746d6c                                html
```

Figura 3.3: Servicio XSDF de la impresora codificado con XBE32

```
(service-id=8e9d7823-d5ac-497c-91d0-fb07ea0c3fb2),
(service-hi-name=Alice's printer),
(service-location-tcp=ipp://169.254.85.139:631,
                      ipp://[fe80::202:b3ff:fe3c:da7a]:631,
                      lpr://169.254.85.139:515,
                      lpr://[fe80::202:b3ff:fe3c:da7a]:515),
(service-location-sctp=ipp://169.254.85.139:631,
                       ipp://[fe80::202:b3ff:fe3c:da7a]:631,
                       lpr://169.254.85.139:515,
                       lpr://[fe80::202:b3ff:fe3c:da7a]:515),
(service-selection-policies=Least Used),
(service-workload=0),
(service-weight=14),
(service-model=Acme Laser Printer 2000),
(service-model-url=http://www.acme.org/printers/lp2000.html),
(color=false),
(duplex=true)
```

Figura 3.4: Servicio XSDF de la impresora codificado con SLPv2

- **Información Adicional del Servicio**: Mientras que con los datos anteriores ya es posible seleccionar y acceder a un servicio automáticamente, la información adicional contiene datos destinados a los usuarios del servicio, como una descripción textual del mismo, o los datos de contacto del administrador del servicio.

Cuando un UA pide información de servicios puede especificar en qué parte de datos está interesado. Esto permite que los UAs no interactivos tan sólo descarguen inicialmente el Estado, la Información Principal y de Localización del servicio, para seleccionar el mejor servidor. Después basta con consultar el Estado para ver si el resto de la información ha cambiado. Todo ello sin descargar la Información Adicional, que no es necesaria para acceder al servicio, y que, al contener mucha información textual para los usuarios, será mucho más voluminosa que el resto. Por el contrario, como en SLP hay una única lista de atributos, es necesario descargar toda la información del servicio para poder acceder al mismo, que puede incluir datos, como por ejemplo una larga descripción textual del mismo.

Los mensajes XSDF, y por tanto la información de servicios, pueden transmitirse por la red empleando cualquier mecanismo de codificación extensible y válido para datos estructurados jerárquicamente, como por ejemplo XML (figura 3.2). Sin embargo, se ha preferido emplear la codificación binaria XBE32 [33], ya que es más compacta, está alineada a 32 bits, y su procesamiento es más eficiente que XML o las reglas de codificación de ASN.1. Por ejemplo, la figura 3.3 muestra cómo se codifica con XBE32 el mismo servicio de impresora de la figura 3.2 (887 bytes) en tan sólo 324 bytes, sin aplicar ningún tipo de algoritmo de compresión, utilizando simplemente una codificación binaria (XBE32) en lugar de textual, como es el caso de XML o SLP (e.g. el servicio SLP de la figura 3.4 ocupa 586 bytes).

Este modelo de servicio es tan extensible que incluso permite representar a los propios Agentes XSDF como servicios. Por el contrario, tanto en SLP como en Rserpool se han definido mensajes y procedimiento específicos para localizar a los Agentes de Directorio y a los Servidores ENRP, puesto que necesitan definir información adicional que no puede incluirse dentro de la información de los servicios. Por tanto en una red XSDF el descubrimiento de servicios es otro servicio de red más y, como tal, puede beneficiarse de sus capacidades de reparto de carga y de localización de servicios, por ejemplo para localizar al Agente de Directorio menos cargado.

3.3.1. Funcionamiento de la cache de servicios

Los Agentes de Servicio generan y mantienen la información de sus servicios, de forma que sólo el SA "hogar" de un servicio puede modificar dicha

información. Sin embargo, para mejorar el rendimiento de la red, los otros dos tipos de Agentes XSDF (UAs y DAs) pueden almacenarla temporalmente en una cache.

Los Agentes de Servicio asignan una duración (*lifetime*) a la información de servicios, que especifica la duración máxima durante la que se espera que esos datos de servicio sigan siendo válidos. Por tanto, la información del servicio puede estar en la cache de los UAs y DAs hasta que se sobrepase ese periodo.

Un Agente de Directorio es simplemente una cache centralizada con la información de los servicios de un grupo. Cuando un SA registra un servicio en el DA empleando XSRP, el SA especifica el *lifetime* esperado de esa información. Cuando un UA pide información de servicios al DA mediante XSLP, la respuesta del DA también contiene la "edad" (*age*) y el tiempo de vida (*ttl*) restante de dicha información.

3.4. Organización en "Reinos"

Al igual que SLP, XSDF emplea el concepto de grupo de trabajo para dividir servicios y usuarios en grupos administrativos, de modo que las búsquedas de los usuarios estén limitadas a los servicios de su mismo grupo. Sin embargo XSDF añade el concepto de "Dominio"[1] para permitir la búsqueda de servicios Remotos. De este modo se define un "Reino"[2] como la combinación de un dominio (si lo hay) y uno o más grupos.

Aunque cada organización puede definir todos los grupos de trabajo que quiera de acuerdo a sus necesidades, XSDF ha definido tres grupos estándar:

- **DEFAULT**: Por defecto todos los servicios pertenecen al grupo DEFAULT y no tienen dominio. Esta configuración por defecto permite que en redes pequeñas sin administración (e.g. SOHO), cualquier dispositivo nuevo que se conecte a la red pueda ser localizado inmediatamente al tener pre-configurada la información de grupo XSDF.

- **LOCAL**: Los servicios que pertenecen a este grupo tan sólo tienen sentido en el contexto del servidor local, como por ejemplo los servicios de gestión como SNMP o Telnet/SSH. Por tanto los servicios de este grupo no deben registrarse en un DA, sino que tan sólo pueden consultarse mediante peticiones enviadas directamente al SA.

[1] Domain, en inglés
[2] Real, en inglés

- **PUBLIC**: Cuando una organización desea hacer públicos alguno de sus servicios, basta con asignarlos al "Reino" formado por su dominio DNS y el grupo PUBLIC para que sean accesibles desde el exterior.

Una de las limitaciones de SLPv2 [39] es que si un grupo está gestionado por varios DA, los SAs deben registrar sus servicios en todos los DAs de su grupo y mantener la información consistente en todos ellos. Por el contrario en XSDF, al igual que en Rserpool, la alta disponibilidad es un requisito fundamental, y la utilización de varios DAs en cada "Reino" será habitual. Por lo tanto, para simplificar la implementación de los SAs y de los UAs XSDF, todos los DAs del mismo grupo se comportan como un único repositorio (gracias a XSSP/XSTP), de modo que aunque los servicios se registran en un sólo DA, estarán accesible desde todos los DAs del grupo.

3.5. Protocolos de XSDF

Aunque XSDF define más protocolos que SLP, en realidad la complejidad para implementarlos no aumenta puesto que los cuatro protocolos son similares entre sí, ya que comparten una estructura de mensajes común [38]. Dicha estructura está formada por una cabecera obligatoria y una o más operaciones que dependen del protocolo en cuestión.

La cabecera de todos los protocolos tiene el mismo formato, e incluye un identificador de transacción para poder asociar preguntas y respuestas, información sobre los Agentes XSDF origen y destino, que se definen como servicios, y el "Reino" al que pertenecen.

En cuanto a las operaciones, éstas pueden ser peticiones, o respuestas a peticiones previas, de forma que cada uno de los cuatro protocolos XSDF define las transacciones y operaciones que necesitan, aunque también a este nivel comparten un gran número de elementos comunes (e.g. operación de error).

La posibilidad de incluir varias operaciones en cada mensaje es otra ventaja de XSDF frente a SLP y Rserpool , ya que, por ejemplo, permite que un SA pueda actualizar el estado de varios de sus servicios en el DA simultáneamente, empleando un único mensaje XSRP.

Además, aunque XSDF emplea UDP como el protocolo de transporte por defecto para reducir la latencia de descubrimiento de servicios y permitir operaciones *multicast*, también es posible utilizar otros protocolo de transporte fiables como TCP o SCTP.

3.5.1. eXtensible Service Location Protocol (XSLP)

XSLP es el protocolo fundamental de la arquitectura XSDF, puesto que es el que permite descubrir los servicios disponibles en la red. Dependiendo de la existencia o no de un Agente de Directorio, los Agentes de Usuario pueden emplear dos modos de funcionamiento de XSLP: consultas *unicast* a un DA, o consultas *multicast* a todos los SAs de la red. Para ello se han definido las siguientes operaciones:

- **Service Request**: Esta operación permite a un UA localizar todos los servicios disponibles de un Tipo determinado (y que ofrezcan el contenido especificado), o servicios particulares identificados por su UUID. Además es posible especificar qué parte de la información se desea recibir, o filtrar servicios ya conocidos (indicando sus UUIDs). Esta operación puede aparecer tanto en mensajes *unicast* dirigidos a un DA o un SA, o en mensajes *multicast*, que recibirán todos los SAs del grupo de trabajo.

- **Service Reply**: Este mensaje sirve de respuesta a la petición anterior, y contiene la información de los servicios seleccionados por la consulta, incluyendo su tiempo de validez máximo (*lifetime*), para que el UA pueda almacenarlo temporalmente en su cache. Además, en caso de que el Tipo de Servicio consultado tenga asociado una política de reparto de carga, el DA ordenará las entradas adecuadamente, seleccionando las mejores.

- **Service Advertisement**: Esta operación es muy similar al Service Reply anterior, puesto que incluye información de servicios, así como su tiempo de validez. Sin embargo en este caso, los SAs o DAs envían este mensaje proactivamente para anunciar servicios importantes. Por ejemplo los propios DAs, se anuncian periódicamente para que el resto de los Agentes XSDF conozcan su información de servicio. Esto permite reducir el número de consultas de localización de DAs, o borrar de la caches de los UAs la información de un servicio inaccesible, poniendo su lifetime a cero.

- **Realm Request**: Esta operación es opcional y permite conocer a qué "Reino" pertenece un Agente XSDF. Los UAs pueden utilizar este mecanismo al inicializarse para descubrir a qué Reino pertenecen, consultando al DA que tengan pre-configurado, por ejemplo mediante DHCP.

- **Realm Reply**: Esta es la respuesta a la petición anterior, y contiene el dominio y los grupos de trabajo a los que pertenece el Agente XSDF que la ha enviado.

- **Service Type Request**: Esta operación también es opcional y, a diferencia del *Service Request*, no solicita información sobre las instancias de un servicio, sino que permite conocer el Tipo de todos los servicios registrados en el SA o DA consultado.

- **Service Type Reply**: Esta respuesta contiene la lista de todos los Tipos de Servicio solicitados mediante la operación anterior. Gracias a esta transacción un administrador de red puede conocer todos los servicios que ofrece su red. Primero preguntando por los Tipos conocidos y luego descubriendo todos los servicios particulares de cada uno.

- **Redirect Reply**: Esta operación puede emplearse como respuesta a cualquiera de las consultas XSLP, e indica que dicha petición debería redirigirse al Agente o Agentes XSDF especificados. Esta operación permite implementar fácilmente políticas de reparto de carga global.

3.5.2. eXtensible Service Registration Protocol (XSRP)

Este protocolo sólo es necesario en redes donde se hayan desplegado Agentes de Directorio, puesto que se emplea para que los Agentes de Servicio registren a sus servicios en el DA, y para que luego actualicen periódicamente su información. Cuando un servicio deja de estar disponible, su SA "hogar" también puede emplear XSRP para de-registrarlo del DA, aunque para ofrecer un funcionamiento robusto los servicios que no sean actualizados por su SA (por ejemplo si éste está inaccesible) también son borrados del repositorio central. Por tanto, XSRP simplemente define tres operaciones de registro y sus respectivas confirmaciones:

- **Service Registration**: Esta operación se emplea para registrar por primera Además, es posible indicar que Políticas de Selección debe aplicar el DA a los servicios de ese Tipo, para ordenarlos en las respuestas a los clientes según su idoneidad.

- **Service Registration Acknowledgement**: Esta respuesta del DA es la confirmación de que el servicio indicado ha sido registrado correctamente en el DA. Además, indica al SA cuándo debe actualizarse dicha información (*minLife-maxLife*) para que el servicio se mantenga activo.

- **Update Service**: Esta petición debe ser enviada periódicamente por los SA para actualizar la información de sus servicios (típicamente el Estado del Servicio), y así evitar que sean borrados del repositorio central.

- **Update Service Acknowledgement**: Esta operación sirve de confirmación a la actualización de información de servicio anterior y, al

igual que el Service Registration Acknowledgement, indica al SA cual es el periodo válido de actualización.

- **Service Deregistration**: Esta petición indica al DA que debe borrar el servicio especificado de todos los grupos de trabajo donde fue registrado, debido a que el servicio va a dejar de estar disponible permanentemente. Si la desconexión es temporal (e.g. si el servicio está saturado y no puede aceptar más peticiones) no es necesario des-registrarlo sino que bastaría con actualizar su Estado con *resources*=0.

- **Service Deregistration Acknowledgement**: Esta es la respuesta a la operación de borrado de servicios anterior.

3.5.3. eXtensible Service Subscription Protocol (XSSP)

Este protocolo es opcional y permite a los Agentes XSDF suscribirse a cambios en la información de los servicios en los que estén interesados. De esta forma no tienen que estar consultando continuamente la información de esos servicios, sino que, una vez suscritos a ella, recibirán un evento cada vez que dicha información cambie. Sin embargo XSSP tan sólo es un protocolo de suscripción, la notificación de estos eventos puede realizarse mediante cualquier otro mecanismo. Por tanto XSSP se limita a suscribir "canales" de notificación de eventos a la información de grupo de servicios.

La gestión de las suscripciones XSSP es muy parecida al registro de servicios y por tanto, tiene un conjunto de operaciones similares a las definidas en XSRP:

- **Subscribe Service**: Esta petición registra en el Agente XSDF destino un canal de notificaciones al que deben enviarse los eventos que afecten a: el "Reino", el Tipo de Servicio, o el servicio individual que se especifique en la petición. Además es posible indicar en qué tipo de eventos se está interesado: Registro de nuevos servicios, Actualización del servicio (distinguiendo si se actualiza información permanente o sólo su Estado), o cuando se De-registra algún servicio de los indicados.

- **Subscribe Service Acknowledgement**: Esta respuesta es la confirmación de que el servicio de notificación anterior ha sido asociado a la información a la que se desea estar subscrito. Además, indica cuando debe actualizarse dicha subscripción mediante el mecanismo de *minLife-maxLife*.

- **Update Subscription**: Esta petición debe ser enviada periódicamente para mantener activa la subscripción registrada previamente. De lo contrario, al igual que ocurría con XSRP, el registro se borrará pasado un tiempo.

- **Update Subscription Acknowledgement**: Esta operación sirve de confirmación a la actualización anterior y, al igual que el Subscribe Service Acknowledgement, indica cual es el intervalo válido para enviar la próxima actualización de la subscripción.

- **Unsubscribe Service**: Esta petición indica al DA que debe borrar la suscripción indicada y desactivar el envío de eventos al servicio de notificación asociado a la misma.

- **Unsubscribe Service Acknowledgement**: Esta es la respuesta a la operación de de-subscripción anterior.

En cuanto a los servicios de notificación de eventos, uno de los protocolos soportados es el propio XSLP, de modo que los cambios en la información del servicio generarán mensajes de Service Advertisement para los Agentes XSDF suscritos. Aunque cualquier Agente XSDF puede suscribirse a la información de servicios de un SA o DA, el uso más habitual de este protocolo será la sincronización de los DAs que gestionan el repositorio de servicios de un Reino común, empleando XSTP como protocolo de notificación.

3.5.4. eXtensible Service Transfer Protocol (XSTP)

El principal objetivo del protocolo XSTP es mantener sincronizada la información de servicios de todos los DAs del mismo Reino. Por ejemplo permite que un DA que está inicializándose pueda obtener de otro DA activo toda la información de servicios conocida, para que pueda empezar a formar parte del repositorio de servicios.

Tal como se comentó en el apartado anterior, además de XSLP, XSTP también puede emplearse como un servicio de notificación de eventos XSSP para que todos los DAs estén subscritos entre sí al "Reino" común. De este modo todas las operaciones XSRP de registro o actualización que reciba cualquiera de ellos puedan aplicarse a todos simultáneamente, por ejemplo empleando multicast.

Este protocolo tan sólo tiene tres operaciones:

- **Service Transfer Request**: Esta operación solicita al DA destino toda la información de servicios que tenga disponible de un Reino determinado. Además, también es posible especificar un *timestamp*, de forma que sólo se envíen los servicios registrados después de ese instante. Esta opción permite re-sincronizar rápidamente el repositorio si el DA se ha desconectado temporalmente, o simplemente puede realizarse esta operación periódicamente para garantizar la consistencia de la información de servicios.

- **Service Transfer Reply**: Esta es la respuesta a la operación de transferencia anterior, y contiene la información sobre todos los servicios solicitados, además de otros datos asociados al registro, como por ejemplo las Políticas de Selección en DA. Todos estos datos deben enviarse a través la conexión TCP/SCTP por la que se recibió la petición Service Transfer Request ya que, a diferencia del resto de operaciones y protocolos de XSDF, esta transacción no emplea UDP como protocolo de transporte, ya que puede requerir que se transfiera una gran cantidad de información.

- **Service Notification**. Esta operación encapsula un mensaje XSRP junto a cualquier otra información relevante acerca del servicio actualizado con XSRP. Por tanto esta operación se emplea para notificar al resto de los DAs que se ha procesado una petición que afecta al registro de servicios común, al que estarán subscritos mediante XSSP.

Capítulo 4

Protocolo XSRP

En este capítulo se van a describir los elementos que componen los distintos mensajes del protocolo XSRP, así como los procedimientos que debe seguir para la realización de las operaciones que realiza: Registrar, Borrar y Actualizar un servicio en el Agente de Directorio.

La fuente de información principal para la realización de este capítulo ha sido el *draft* de XSRP[35].

El eXtensible Service Registration Protocol (XSRP) es un protocolo ligero que permite a los Agentes de Servicio registrar información de servicios en *Reinos* gestionados por uno o varios Agentes de Directorio. Gracias al mismo, los Agentes de Usuario podrán obtener esta información de servicios de los Agentes de Directorio de una manera más eficiente que con búsquedas *multicast*.

4.1. Introducción

XSDF es un entorno integrado para el descubrimiento de servicios y el reparto de carga. Esto permite a los usuarios encontrar dinámicamente los servicios disponibles en la red y elegir el más adecuado siguiendo ciertas políticas de selección. En XSDF, los Agentes de Usuario pueden enviar *Service Request multicast* directamente a los Agentes de Servicio. En redes más extensas, los UAs preguntan por información de servicios a un repositorio central gestionado por el llamado Agente de Directorio.

XSRP es el protocolo empleado por los SAs para registrar información de los servicios en los DAs que gestionan el Ámbito de servicios. Además, cuando se registra un servicio vía XSRP, un SA puede mandar a un DA aplicar alguna política de selección de servicios del mismo tipo. Por tanto, cuando un DA responde un XSLP Request de un UA, puede clasificar servicios u ofrecer un subconjunto de ellos.

4.1.1. Definiciones

- ***Domain* - Dominio**: Un Dominio es una organización administrativa que contiene Usuarios y Servicios
- ***Scope* - Ámbito**: Un Ámbito es un subconjunto de Usuarios y Servicios de un Dominio, que típicamente forman un grupo de trabajo.
- ***Realm* - Reino**: Un Reino es una combinación de un Dominio y uno o más "Ámbitos" de cada dominio
- **Servicio**: Un Servicio es cuaquier proceso de red que proporciona algún tipo de funcionalidad para los Usuarios u otros Servicios. Un Servicio pertenece a un único Reino.
- ***User Agent* (UA) - Agente de Usuario**: Es un proceso que trabaja del lado del usuario para descubrir servicios y seleccionar el mejor entre los disponibles. Los UAs obtienen información del servicio de los Agentes de Directorio y Agentes de Servicio vía XSLP.
- ***Service Agent* (SA) - Agente de Servicio**: Un Agente de Servicio es un Servicio en sí mismo. Los Agentes de Usuario preguntan a los SAs por Servicios vía XSLP, y/o registrar información de Servicios en Agentes de Directorio vía XSRP.
- **Directory Agent (DA) - Agente de Directorio**: Un Agente de Directorio es un servicio que, cuando se despliega, actúa como un repositorio central de toda la información de Servicios de un Reino. Los Agentes de Servicio registran sus Servicios en un DA utilizando el protocolo XSRP, de modo que los Agentes de Usuario pueden preguntar por esa información mediante XSLP. En un Ámbito puede haber varios DAs, los cuales se coordinan mediante los protocolos XSSP y XSTP.
- **Agente XSDF**: Cualquier proceso que emplea algún protocolo XSDF para interactuar con otro Agente XSDF. Dependiendo del protocolo, un Agente XSDF puede comportarse como cliente ó como servidor.
- **Agente XSRP**: Un Agente XSRP es un Agente XSDF que implementa el protocolo XSRP (i.e. un SA o DA). Los SAs se comportan como clientes, mientras que los DAs son servidores XSRP.

4.2. Formato de Mensajes

Esta sección define el formato de todos los mensajes XSRP. Estos mensajes se intercambian entre un Agente de Servicio y un Agente de Directorio.

Los mensajes del protocolo XSDF están codificados utilizando XBE32 [33]. Las definiciones de los elementos y atributos comunes a todos los protocolos XSDF pueden encontrarse en [38].

4.2.1. Elemento XSRPv1

Todos los mensajes XSDF tienen un formato común: un solo elemento XBE32 que contiene una cabecera y una o más operaciones del protocolo. Los mensajes XSRP están codificados dentro de un elemento XSRPv1.

Elemento complejo XSRPv1 (0x0141)

```
<xsrpv1>
   <header>
     <xid/>
     <realm>
       <domain/>
       <scope/>
     </realm>
     <source>
       <id/>
     </source>
     <destination>
       <id/>
     </destination>
   </header>
   <XSRP Operation #1/>
   <XSRP Operation #2/>
</xsrpv1>
```

Los elementos de *header* están definidos en [38]. Las siguientes subsecciones describen los elementos que codifican las operaciones XSRP.

4.2.2. Elemento *Register Service*

Un SA envía una petición de esta operación a un DA para registrar el servicio especificado en los distintos Ámbitos gestionados por el DA.

Elemento complejo *Register Service* (0x142)

```
<registerService>
   <realmTarget>
     <scopesMap/>
   </realmTarget>
   <service>
```

```
      <id/>
      <serviceState/>
      <serviceMainInfo/>
      <serviceNetInfo/>
      <serviceAddInfo/>
   </service>
   <registerState/>
   <registerInformation/>
   <updateInformation>
     <minLife/>
     <maxLife/>
   </updateInformation>
</registerService>
```

Los elementos *Realm Target*, *Service* y *Update Information* están definidos en [38]. El Estado de Registro y los elementos de Información están definidos en la sección 4.2.9.

El elemento *Realm Target* puede contener un subconjunto de Reinos y Ámbitos de la cabecera donde los servicios pueden ser registrados. Si el *Realm Target* está vacío, el servicio debe ser registrado en todos los Ámbitos especificados en la cabecera del mensaje.

Cuando un servicio es registrado por primera vez, el elemento *Service* tiene que contener la siguiente información: Id, *Service State*, *Service Main Information*, *Service Location* y *Service Additional Information*.

Los SAs tienen que especificar algunas propiedades cuando registran un servicio. El elemento *Register Information* tiene que contener el tiempo de vida especificado en la información del servicio. Los elementos *Selection State* e *Information* pueden estar también incluidos. En este caso, el SA solicita a un DA que aplique una política de selección para el tipo de servicio especificado.

Los SAs pueden incluir un elemento *Update Information* para sugerirle al DA cual es el intervalo de refresco apropiado para este nuevo servicio.

4.2.3. Elemento complejo *Register Service Acknowledgment* (0x0143)

El DA envía esta operación para confirmar una operación "registerService" previa enviada por un SA.

```
<registerServiceAck>
   <realmTarget>
```

```
      <scopesMap/>
    </realmTarget>
    <service>
       <id/>
    </service>
    <updateInformation>
      <minLife/>
      <maxLife/>
    </updateInformation>
 </registerServiceAck>
```

Realm Target, *Service* y *Update Information* están definidos en [38].

El *Realm Target* tiene que contener un subconjunto de Reinos y Ámbitos de la cabecera donde los servicios han sido registrados. Si el *Realm Target* está vacío, el servicio ha sido registrado en todos los Ámbitos especificados en la cabecera del mensaje.

El elemento *Service* tiene que contener el Id del servicio actualizado. Otra información del servicio es opcional y puede no aparecer.

En el elemento *Update Information*, el DA define el intervalo de tiempo en el que un SA debe refrescar el registro del servicio con otro *updateService*. De lo contrario, el servicio será borrado después de un tiempo "maxLife".

4.2.4. Elemento complejo *Update Service* (0x0144)

Un SA envía esta operación para refrescar la información del servicio registrado en el DA que gestiona el Ámbitos. Esta operación también puede actualizar las propiedades del registro.

```
 <updateService>
    <realmTarget>
      <scopesMap/>
    </realmTarget>
    <service>
       <id/>
       <serviceState/>
       <serviceMainInfo/>
       <serviceNetInfo/>
       <serviceAddInfo/>
    </service>
    <registerState/>
    <registerInformation/>
```

```
    <updateInformation>
      <minLife/>
      <maxLife/>
    </updateInformation>
 </updateService>
```

Realm Target, *Service* y *Update Information* están definidos en [38]. Los elementos *Register Information* y *Register State* están definidos en la sección 4.2.9.

El *Realm Target* puede contener un subconjunto de Reinos y Ámbitos de la cabecera donde los servicios han sido registrados. Si el *Realm Target* está vacío, el servicio debe ser actualizado en todos los Ámbitos especificados en la cabecera del mensaje.

El elemento *Service* tiene que contener el Id y el elemento State de un servicio registrado. El SA puede incluir otra información del servicio para actualizar el servicio ya registrado. En este caso, los atributos apropiados en el elemento "metaInfo" dentro del elemento *Service State* tienen que incrementarse.

Los elementos *Register Information* y *Register State* también pueden incluirse para actualizar las propiedades de registro. Los elementos *Selection State* y *Selection Information* no pueden ser incluidos, a no ser que se hubiese asignado una política de selección cuando un servicio fue registrado por primera vez. En este caso, la política de selección tiene que ser la misma que la registrada previamente.

Los SAs pueden incluir un elemento *Update Information* para sugerir a los DAs cual es el intervalo de refresco adecuando para ese servicio. No obstante, un SA tiene que cumplir con el intervalo de actualización propuesto por el DA en su respuesta, sin importar la sugerencia.

4.2.5. Elemento complejo *Update Service Acknowledgment* (0x0145)

El DA envía esta operación para confirmar un "updateService" previo realizado por un SA.

```
<updateServiceAck>
   <realmTarget>
     <scopesMap/>
   </realmTarget>
   <service>
```

```
      <id/>
    </service>
    <updateInformation>
      <minLife/>
      <maxLife/>
    </updateInformation>
 </updateServiceAck>
```

Realm Target, *Service* y *Update Information* están definidos en [38]

El elemento *Realm Target* tiene que contener un subconjunto de los Reinos y Ámbitos de la cabecera donde los servicios han sido registrados. Si el *Realm Target* está vacío, los servicios han sido actualizados en todos los Ámbitos especificados en la cabecera del mensaje.

El elemento *Service* tiene que contener el Id del servicio actualizado. Otra información de servicio es opcional y puede no aparecer.

En el elemento *Update Information*, el DA define el intervalo de tiempo en el que un SA debe refrescar el registro de un servicio con otro "updateService". De lo contrario, el servicio será borrado al pasar un tiempo "maxLife".

4.2.6. Elemento complejo *Deregister Service* (0x0146)

Con esta operación, un SA se comunica con un DA para borrar el servicio especificado de los Ámbitos donde esté registrado.

```
 <deregisterService>
    <realmTarget>
      <scopesMap/>
    </realmTarget>
    <service>
       <id/>
    </service>
 </deregisterService>
```

Los elementos *Realm Target* y *Service* están definidos en [38]

El elemento *Realm Target* puede contener un subconjunto de los Ámbitos y Reinos de la cabecera donde están registrados los servicios. Si el *Realm Target* está vacío, el servicio debe ser borrado de todos los Ámbitos especificados en la cabecera del mensaje.

4.2.7. Elemento complejo *Deregister Service Acknowledgment* (0x0147)

El DA envía esta operación para confirmar una operación "deregisterService" previa enviado por un SA.

```
<deregisterServiceAck>
   <realmTarget>
     <scopesMap/>
   </realmTarget>
   <service>
      <id/>
   </service>
</deregisterServiceAck>
```

Los elementos *Realm Target* y *Service* están definidos en [38].

El elemento *Realm Target* tiene que contener un subconjunto de los Ámbitos y Reinos de la cabecera donde están registrados los servicios. Si el *Realm Target* está vacío, el servicio ha sido borrado de todos los Ámbitos especificados en la cabecera del mensaje.

El elemento *Service* tiene que contener el Id del servicio borrado. Otra información de servicio es opcional y puede no aparecer.

4.2.8. Elemento Error

XSDF define un elemento simple de Error para todos los tipos de errores del protocolo. Véase en [38] el formato del elemento Error y una descripción más detallada de los errores comunes XSDF.

Un error XSRP se codifica dentro de un elemento de Error que contiene el código de Error, el nombre de los atributos, y elementos de descripción opcionales. Todos los errores XSRP tienen que contener un elemento *Service* adicional con el Id del servicio que ha causado dicho error. Otra información del servicio es opcional y puede no aparecer.

El error INCOMPATIBLE_POLICY puede contener también el elemento *Selection Information*, asociado con el servicio registrado, que está en conflicto con el solicitado

4.2.9. Parámetros Comunes de Operación

Cuando un servicio se registra y después se actualiza, el SA puede especificar los datos del servicio, además de cierta información de registro. En esta sección se define la información de registro que puede aparecer en las peticiones "registerService" y "updateService".

Elemento complejo *Register State* (0x0148)

Este elemento especifica las propiedades volátiles del registro de un servicio. El *Register State* tiene que estar incluido en las peticiones "registerService" (incluso si está vacío) y puede ser actualizado después con un "updateService".

```
<registerState>
   <selectState>
     <resources/>
     <workload/>
   </selectState>
</registerState>
```

El elemento *Selection State* está definido en [38]. Contiene la información volátil para el proceso de selección de DA asociado a un cierto tipo de servicio.

Elemento complejo *Register Information* (0x0149)

Este elemento especifica las propiedades no-volátiles del registro de un servicio. El *Register Information* tiene que estar incluido en las peticiones "registerService" y algunos datos pueden ser actualizados más tarde por mensajes "updateService".

```
<registerInformation>
   <cacheInformation>
      <lifetime/>
   </cacheInformation>
   <selectInformation>
      <policies/>
      <priority/>
      <weight/>
   </selectInformation>
</registerInformation>
```

El elemento *Selection Information* está definido en [38].

El elemento *Selection Information* puede estar incluido en el *Service Registration*. En este caso, el SA solicita al DA que aplique la política de selección especificada a los servicios del mismo tipo que el registrado. Toda la información de selección, excepto la política de selección, puede actualizarse después, aunque sólo si estaba registrada inicialmente.

Elemento complejo *Cache Information* (0x014A)

Este elemento contiene el tiempo de vida esperado de la información de servicio asociada. Tiene que estar especificado en el registro del servicio y puede ser actualizado después, cuando se incluye dentro de peticiones "updateService".

```
<cacheInformation>
    <lifetime/>
</cacheInformation>
```

Este atributo contiene el tiempo máximo, expresado en milisegundos, durante el cual la información del servicio puede considerarse válida.

4.3. Prodecimientos Operativos de XSRP

Todos los agentes XSDF tienen que seguir los procedimientos comunes para la generación de mensajes, transmisión y proceso defindos en [38]. Esta sección especifica los procedimientos específicos para los agentes XSRP.

Todos los SAs y DAs tienen que implementar XSRP. No obstante, como los DAs no son entidades obligatorias dentro del entorno XSDF, en algunos escenarios XSRP puede no ser necesario. Esta sección sólo se refiere a la interacción entre SA y DA. Para la interacción entre UA y SA así como de UA y DA consúltese [34].

Todos los servicios tienen que tener un UUID [26] de 128 bits único en el Dominio al que pertenece. Cada servicio puede pertenecer a un Reino con múltiples Ámbitos. Un servicio siempre tiene que pertenecer al mismo Reino. Por lo tanto, los SAs tienen que actualizar y borrar la información de servicio de todos los Ámbitos donde estaba registrado. Así mismo, la información de servicio tiene que estar controlada por un único SA, llamado "Home SA", para asegurarse de que la información de servicio tiene que ser la misma para todos los Ámbitos a los que el servicio pertenece.

XSRP es un protocolo Cliente-Servidor. El SA actúa como cliente, enviando peticiones, mientras que el DA actúa como un servidor, procesando esas peticiones y enviando respuestas.

Todos los agentes XSRP tienen que soportar TCP como protocolo de transporte por defecto, aunque también emplear UDP ó SCP. El número de puerto por defecto para todos los protocolos de transporte es 721.

4.3.1. Inicialización de *Service Agent*

Como cualquier otro agente XSDF, los SAs tienen que estar configurados con el Reino al que pertenecen, y obtener el DA que gestiona esos Ámbitos, en caso de haberlo. Por lo tanto, tienen que emplear los procedimientos de inicialización descritos en [38].

Cuando un agente de servicio se inicia, éste tiene que generar un identificador único de servicio de 128 bits. Los SAs y DAs tienen que estar configurados con el Reino al que pertenecen. Todos los servicios gestionados por un agente XSRP tienen que pertencer al Reino o a un subconjunto de los Ámbitos del Reino.

Puede ocurrir que un SA pertenezca a un Reino que no esté gestionado por un DA. En este caso, debe comportarse como un *Stand-Alone SA* para los servicios de ese Reino, y responder a las peticiones XSLP de los UAs. Sin embargo, el SA tiene que seguir escuchando los anuncios de nuevos DAs del Reino para empezar su proceso de registro. Lo mismo puede pasar si todos los DAs existentes en el Reino no estuviesen operativos: Los SAs pueden comportarse como *Stand-Alone* hasta que se descubra un DA en el Reino.

En el modo *Stand-Alone*, los SAs tienen que unirse a los grupos *multicast* IPv4 y IPv6 apropiados como se define en [38], para los servicios que pertenecen a los Ámbitos que no están gestionados por ningún DA. Particularmente, deben unirse a los grupos *multicast* asociados con el tipo de servicio “xsdf:sa”.

Cuando un SA conoce uno o más DAs en los Reinos compatibles que soportan el protocolo XSRP; ese SA tiene que registrar, y posteriormente actualizar, todos los servicios pertenecientes a esos Reinos. No obstante, los servicios del Ámbito “LOCAL” están restringidas al servidor local. Por lo tanto, no pueden ser gestionados por los DAs ni registrados en ellos.

Los SAs tienen que esperar un tiempo aleatorio, entre 0 y REGISTRATION_DELAY antes de registrar los servicios en un nuevo Reino (i.e. por el descubrimiento de un DA en un Reino anteriormente *Stand-Alone*). Adicionalmente, los anuncios de servicios de los DAs pueden incluir un elemento *Update Information* dentro del elemento XSRP. En este caso, los SAs tienen que esperar un intervalo de tiempo aleatorio entre los atributos "minLife" y "maxLife" antes de actualizar la información.

Como un SA es en sí mismo un servicio, los SAs pueden registrar su información de servicio en los Reinos a los que pertenecen. El atributo *Type* para la información de servicio del SA es "xsrp:sa".

La fase de iniciación acaba cuando el SA ha registrado todos los servicios en el DAs de los Reinos a los que pertenece, y se comporta como un SA *Stand-Alone* para los servicios en Ámbitos no gestionados.

4.3.2. Registro de un Servicio

Los SAs registran sus servicios en un Reino enviando una petición "registerService" a cualquier DA de su Reino. Varios servicios pueden registrarse a la vez incluyendo múltiples peticiones "registerService" dentro de un único mensaje XSRP.

Una vez que se genera el mensaje XSRP con la operación "registerService", se envía al DA. Si un mensaje no puede entregarse, debe escogerse el siguiente DA del mismo Reino. Si no hay más DAs, se aborta el proceso de registro y el SA vuelve al modo *Stand-Alone* para ese Reino.

Cuando un DA recibe un mensaje XSRP válido con varias operaciones "registerService" autorizadas, se tienen que seguir los siguientes pasos para procesar la operación:

1. Obtener la lista de los Ámbitos donde se va a registrar el servicio del elemento *Target* de la petición. Si está vacío, debe cogerlo de la cabecera del mensaje. Dichos Ámbitos tienen que aparecer en la cabecera del mensaje. De lo contrario, el procesado del mensaje debe abortarse.

2. Comprobar que no hay otro servicio con el mismo Id ya registrado en el mismo Ámbito. En este caso, la operación debe ser interrumpida y se crea un error SERVICE_COLLISION.

3. Buscar un servicio registrado con el mismo tipo que el que se quiere registrar. Si hay otro servicio, pero con una política de selección incompatible, la operación debe ser interrumpida y se crea un error INCOMPATIBLE_POLICY

4. Si no se ha producido ningún error, el DA tiene que registrar el servicio, con la información de servicio especificada y las propiedades del registro, en todos los Ámbitos de la lista.

5. Elegir un intervalo de actualización y enviar una respuesta "registerServiceAck". La informacion de actualización sugerida puede estar incluida en la respuesta.

6. Cambiar el LAST_UPDATE_TIMESTAMP asociado y las variables del HOME_SA con el tiempo actual de registro y el Id del servicio del SA del mensaje origen, respectivamente.

7. Cambiar el MAX_LIFE_TIMER asociado al nuevo servicio registrado. El temporizador tiene que expirar después de un tiempo "maxLife".

Para cada petición "registerService", el DA tiene que generar un "registerServiceAck" si el servicio ha sido registrado, o un elemento de error en caso de haberse producido alguno. Ambas respuestas tienen que incluir el Id del servicio registrado.

Cuando se recibe un mensaje XSRP válido que contiene una o varias respuestas "registerServiceAck". Los SAs tienen que revisar los atributos de tiempo de vida mínimo y máximo y cambiar el UPDATE_TIMER asociado al servicio registrado. El tiempo de expiración tiene que estar entre "minLife" y "maxLife" milisegundos.

A fin de que los SAs sepan qué política de selección soporta un DA, dicha información puede incluirse en un elemento del protocolo XSRP [38]. Concretamente, un elemento *Selection Information* con la política de selección soportada.

4.3.3. Actualización de un Servicio

Cuando expira un UPDATE_TIMER asociado a un servicio, el SA debe actualizar el servicio en todos los Ámbitos donde el servicio está registrado. Por lo tanto, tiene que enviar peticiones "updateService" a los DAs que gestionan esos Ámbitos.

La información de servicio puede actualizarse con el mismo mecanismo en cualquier momento. Sin embargo, las actualizaciones de información deben retrasarse hasta que haya pasado un tiempo "minLife" desde la última actualización.

Una vez que se ha generado un mensaje XSRP con la operación "updateService", ésta se envía al DA. Si el mensaje no puede entregarse, debe elegirse el siguiente DA dentro del mismo Reino. Si no hay más DAs, el proceso de registro se aborta y el SA vuelve al modo *Stand-Alone* para ese Reino.

Cuando un DA recibe un mensaje XSRP válido con varias operaciones "updateService", debe seguir los siguientes pasos para procesar dicha operación:

1. Obtener la lista de los Ámbitos donde se va a registrar el servicio del elemento *Target* de la petición. Si está vacío, debe cogerlo de la cabecera del mensaje. Todos los Ámbitos tienen que aparecer en la cabecera del mensaje, de lo contrario el procesado del mensaje debe abortarse.

2. Buscar el registro del servicio que va a ser actualizado en los Ámbitos que gestiona el DA. Si no se encuentra ningún servicio con el Id especificado, la operación tiene que ser interrumpida y se crea un elemento de error SERVICE_NOT_FOUND.

3. Comprobar que el Id del servicio del mensaje del SA origen sea igual que la variable HOME_SA. En otro caso, la operación tiene que ser interrumpida y se genera un error INVALID_HOME_SA.

4. Si la petición de actualización contiene un elemento *Selection Information*, hay que comprobar que el servicio ha sido registrado con la misma política de selección. En otro caso, la operación debe ser interrumpida y se crea un error INCOMPATIBLE_POLICY.

5. Si no ha ocurrido ningún error, el DA tiene que actualizar el servicio en todos los Ámbitos en los que está registrado. La información específica del servicio y las propiedades del registro tienen que ser actualizadas, en caso de haberlas. El atributo *State TimeStamp* de ambos elementos *Service State* tiene que ser comparado antes de que se lleve a cabo cualquier actualización para comprobar que información es la más reciente.

6. Elegir un intervalos de actualización y crear una respuesta "updateServiceAck". La información de actualización sugerida por el SA puede tomarse en cuenta.

7. Se asocia el tiempo actual a la variable de última actualización LAST_UPDATE_TIMESTAMP.

8. Reestablecer MAX_LIFE_TIMER asociado con el servicio actualizado. Este temporizador tiene que expirar después de un tiempo "maxLife", a menos que se actualice antes, tal como se informa en la respuesta "updateServiceAck"

Para cada petición "updateService", el DA tiene que generar un "updateServiceAck" si el servicio ha sido actualizado, o un elemento Error si ha ocurrido alguno. Ambas respuestas tienen que incluir el Id del servicio implicado en el proceso.

Cuando se recibe un mensaje XSRP válido que contiene una o más respuestas "updateServiceAck", el SA tiene que comprobar los atributos tiempo de vida mímino y máximo y reestablecer el temporizador UPDATE_TIMER asociado al servicio registrado. El tiempo para que expire este temporizador tiene que estar distribuido uniformemente entre "minLife" y "maxLife" milisegundos.

4.3.4. Deregistro de un Servicio

Cuando se deja de ofrecer un servicio de red o éste no está disponible, su "Home SA" tiene que borrarlo de todos los Reinos en los que está registrado, empleando peticiones XSRP "deregisterService". Se pueden borrar varios servicios en un mismo mensaje, incluyendo en éste varias operaciones "deregisterService".

Una vez el mensaje XSRP ha sido generado con la operación "deregisterService", éste se envía al DA. Si un mensaje no puede ser entregado, se debe escoger al siguiente DA del mismo Reino. Si no hay más DAs, el proceso de registro es abortado y el SA vuelve al modo *Stand-Alone* para ese Reino.

Cuando un DA recibe un mensaje XSRP válido con una o varias operaciones "deregisterService", tiene que seguir los siguientes pasos para procesar cada operación:

1. Obtener la lista de los Ámbitos donde se va a registrar el servicio del elemento *Target* de la petición. Si está vacío, se coge de la cabecera del mensaje. Estos Ámbitos tienen que haber sido listados en la cabecera del mensaje. De lo contrario, el procesado del mensaje debe ser abortado.

2. Buscar el servicio que va a ser borrado en los Ámbitos. Si no se encuentra ningún servicio con el Id especificado, la operación tiene que ser interrumpida y se crea un elemento de error SERVICE_NOT_FOUND.

3. Comprobar que el Id del servicio del mensaje del SA origen sea igual que la variable HOME_SA. En otro caso, la operación tiene que ser interrumpida y se genera un error INVALID_HOME_SA.

4. Si no ha ocurrido ningún error, el DA tiene que borrar el servicio especificado de todos los Ámbitos en los que ha sido registrado, borrando

la información de servicio, las propiedades del registro, las variables asociadas a la caché y limpiar el temporizador MAX_LIFE_TIMER

Para cada petición "deregisterService", el DA tiene que generar un "deregisterServiceAck" si el servicio ha sido borrado, o un elemento Error si ha ocurrido alguno. Ambas respuestas tienen que incluir el Id del servicio implicado en el proceso.

Cuando el temporizador MAX_LIFE_TIME asociado al registro de un servicio expira, el DA tiene que borrar ese servicio de todos los Ámbitos en los que está registrado. La información del servicio y las propiedades del registro tienen que ser borrados de la cache.

4.4. Consideraciones de seguridad

XSDF se basa en los mecanismos de seguridad proporcionados por los niveles inferiores, como TLS o IPSec. Sin embargo, define varios mecanismos de autenticación que pueden ser empleados si no los proporcionan los niveles inferiores. Véase [38] para una descripción más detallada de estos mecanismos.

XSRP sobre TLS se denomina "xsrp/tls" y su número de puerto por defecto para los protocolos de transporte TCP y SCTP es 722.

Nótese que comprobar la variable HOME_SA intenta evitar colisiones y configuraciones erróneas. Esto puede no considerarse como un mecanismo de seguridad ya que el Id del servicio del SA puede ser obtenido y falseado fácilmente.

Capítulo 5

Diseño de un Cliente-Servidor XSRP

En este capítulo se pretende explicar el diseño de los distintos agentes que componen el protocolo XSRP. Puesto que ya existen unas implementaciones previas de XBE32 [42] y XSLP [41], se explicará desde donde se parte: librerías, clases, etc. y se expondrán los diferentes casos de uso de la comunicación entre un SA y un DA empleando el protocolo XSRP.

5.1. Arquitectura Inicial

Para la realización del diseño de XSRP, en primer lugar es necesario realizar un análisis de la arquitectura XSDF existente, para saber donde encaja nuestra aportación y poder realizar un diseño adecuado a las condiciones de partida.

Estas condiciones de partida son resultado de la realización de dos proyectos fin de carrera realizados previamente. El primero nos proporciona la librería XBE32 [42] y el segundo la implementación del protocolo XSLP [41].

Para estudiar la implementación realizada, en primer lugar es necesario comentar la arquitectura de la misma. Se ha realizado una estructura por capas, la cual hace que cada una de las mismas se encarga de una funcionalidad específica.

La estructura sería la siguiente en el lado del servidor:

SAServer
XSLPServer
XSDFCommon
XBE32

La capa XBE32 siendo la parte inferior de la arquitectura es la encargada de la codificación y decodificación de los mensajes XBE32 recibidos. La capa XSDFCommon es la encargada del transporte de los mensajes, en este caso al tratarse del lado del servidor, será la encargada de la recepción de las peticiones y envío de las respuestas. La capa XSLPServer es la encargada del tratamiento de los mensajes XSLP recibidos y la creación de la respuesta. La capa superior, SAServer, tiene un funcionamiento asíncrono. Éste se registra a los eventos del XSLPServer, para la realización de las operaciones pertinentes. La capa XSLP se encarga de implementar el protocolo en sí, mientras que la lógica del servidor (e.g. consultar la cache) está en el SAServer.

De modo análogo al lado del servidor, tenemos una estrutura en capas en el lado del cliente. La estructura sería la siguiente:

UAClient
XSLPClient
XSDFCommon
XBE32

Las dos capas inferiores realizan la misma función que en el caso anterior, pero del lado del cliente. Es decir, en este caso la capa XBE32 realiza la codificación de los mensajes XBE32 antes de su envío. La capa inmediatamente superior tiene la misma función comentada en el caso anterior, transporte de los mensajes, y al tratarse del lado del cliente sería concretamente el envío y posterior recepción de los mismos. La capa del XSLPClient se encarga de formar el mensaje, que será enviado utilizando las capas inferiores. La parte superior, el UAClient, es el encargado de implementar el API del *User Agent*.

En el cuadro que se muestra a continuación se pueden ver los métodos del API del Agente de Usuario, incluyendo los parámetros que se reciben y se devuelven.

API *userAgent*	
void	connect()
void	disconnect()
Realm	getRealm()
ServiceType[ ]	getServiceTypes()
ServiceType[ ]	getServiceTypes(Realm realm)
Iterator<Service>	getServices (ServiceType type)
Iterator<Service>	getServices (Realm realm, ServiceType type)
Service	getFullService (ServiceType type, UUID id)
Service	getFullService (Realm realm, ServiceType type, UUID id)
XSDFConfiguration	getXSDFConfiguration()

Los métodos `connect()` y `disconnect()` indicarán si se trata de una comunicación orientada a conexión o no orientada a conexión respectivamente. Los métodos `getServiceTypes()` y `getServiceTypes(Realm realm)` devuelven todos los tipos de los servicios registrados en un `realm`. Los métodos `getServices (ServiceType type)` y `getServices (Realm realm, ServiceType type)` devuelven los servicios registrados del tipo determinado. Los métodos `getFullService(ServiceType type, UUID id)` y `getFullService(Realm realm ServiceType type, UUID id)` devuelven un objeto de la clase `Service` que encapsula un servicio de ese tipo y con ese identificador perteneciente al `realm` en cuestión. Por último, el método `getXSDFConfiguration()` devuelve un objeto `XSDFConfiguration` que encapsula la configuración del agente XSDF. Siempre que no se facilite el parámetro `realm`, se obtiene mediante el método `getRealm()` que proporciona este mismo API.

Como se ha comentado, en el primer proyecto fin de carrera se desarrolló la librería XBE32 que se encarga de la codificación y decodificación de los distintos mensajes intercambiados por los agentes. Antes de ser enviado, el mensaje se codifica utilizando los métodos proporcionados por esta librería. Una vez es recibido, se le aplica un *parser* para identificar cada uno de los campos que componen el mensaje y se realiza la decodificación del mismo.

En la figura 5.1 podemos apreciar el diagrama de clases que componen la arquitectura de la librería XBE32. Se ha optado por mostrar únicamente las clases más relevantes y las relaciones entre las mismas, sin incluir los atributos y métodos de cada una de las clases, ya que sería ilegible y entorpecería la correcta comprensión de la estructura.

Para lograr los objetivos funcionales de la librería XBE32 se dispone de un conjunto de clases que se encargan de los diferentes procesos necesarios para el tratamiento de los mensajes XBE32. Las clases XBE32Parser,

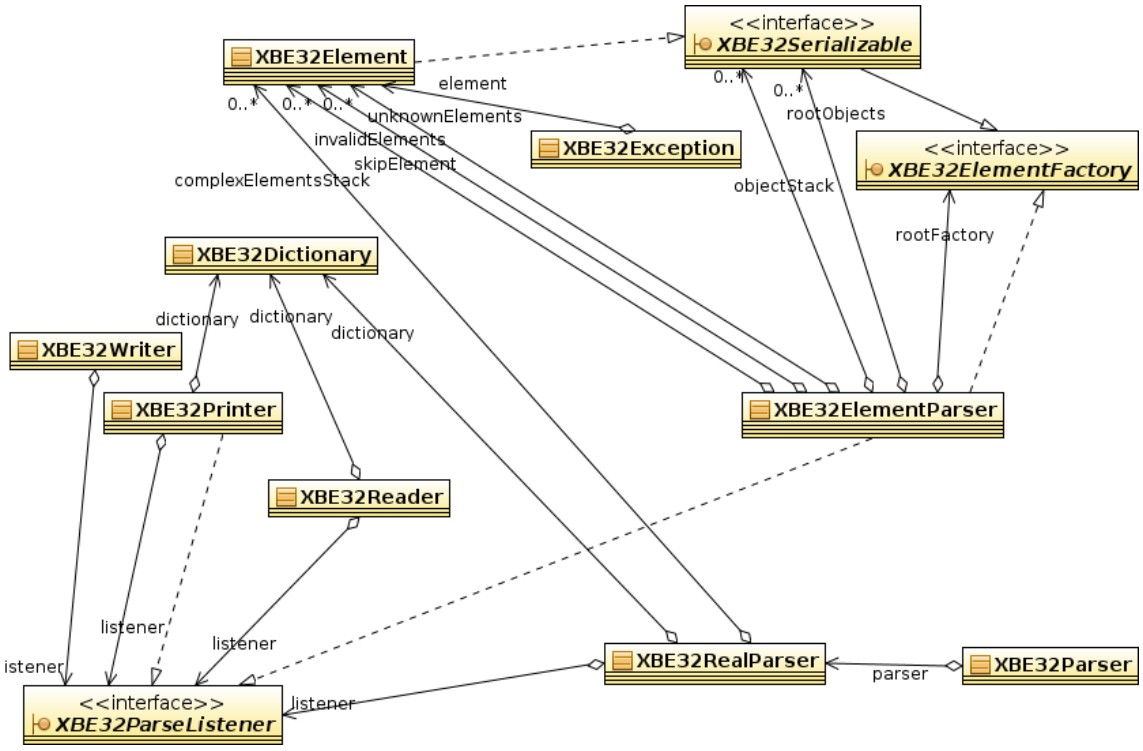


Figura 5.1: Diagrama de clases del paquete XBE32

XBE32Element y XBE32ElementParser permiten de manera sencilla la extracción de información contenida en un flujo de datos XBE32 con el fin de abstraer a las capas superiores de este proceso. Concretamente la clase `XBE32Parser` ofrece un interfaz asíncrono parecido a *SAX* donde se recibe un evento por cada elemento XBE32 procesado. Mientras que `XBE32ElementParser` permite crear un arbol de `XBE32Element` similar al árbol DOM en XML. Por otro lado, las clases `XBE32Reader` y `XBE32Printer`, son las encargadas del proceso de codificación y decodificación de un fichero XML a XBE32.

Además de la librería XBE32 disponemos del paquete de clases XSDFCommon que se encargará de la función de transporte de los mensajes dentro de la arquitectura de XSDF.

En la figura 5.2 podemos ver el diagrama de clases del paquete XSDFCommon. Al igual que en el caso anterior, se ha prescincido de mostrar los métodos de las clases mostradas con el fin de aumentar legibilidad del diagrama. En este conjunto de clases consideramos relevante realizar una breve explicación de las más significativas. La clase `XSDFMessage` modela la información de los mensajes XSDF. De ella heredarán las clases encargadas de

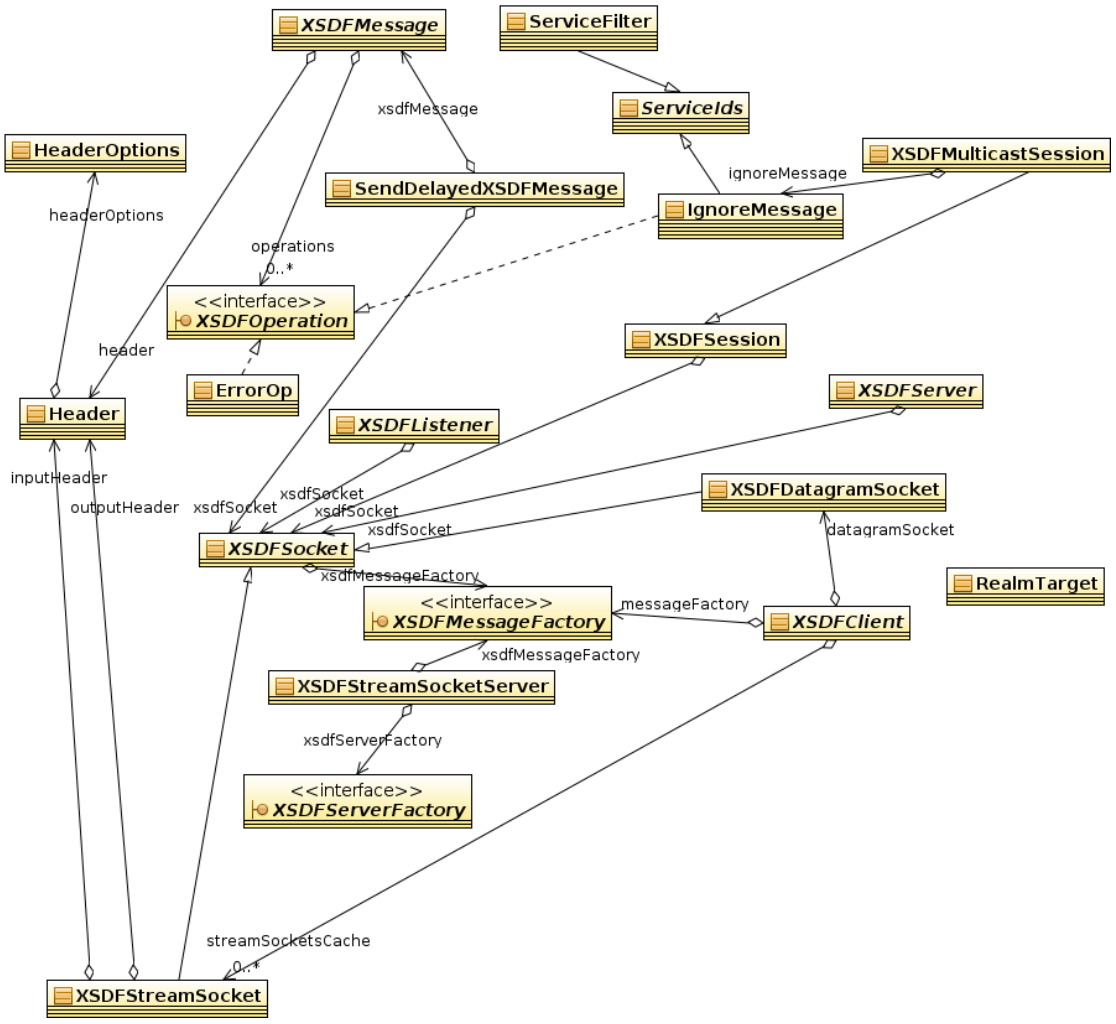


Figura 5.2: Diagrama de clases del paquete XSDFCommon

encapsular mensajes de los distintos protocolos. La clase `XSDFSession` es la encargada de encapsular la información necesaria para realizar el intercambio de mensajes como es el `XSDFSocket`, el protocolo de XSDF empleado en la sesión, el protocolo de transporte que será utilizado para realizar la comunicación, entre otros parámetros. Por otro lado la clase `XSDFSocket` es la superclase de todos los sockets XSDF que proporciona un API común para el envío, recepción e intercambio fiable de los mensajes XSDF. A partir de dicha clase, sus subclases `XSDFDatagramSocket` y `XSDFStreamSocket`, permiten intercambiar mensajes XSDF en segmentos UDP y conexiones TCP respectivamente.

Además de estos dos paquetes que nos proporcionan la capacidad de codificación y decodificación de los distintos mensajes como el transporte de los mismos, disponemos de un protocolo ya implementado, XSLP, el cual es el encargado de realizar los procedimientos necesarios de descubrimiento de servicio. En este caso, XSRP necesitará la funcionalidad aportada por XSLP para encontrar los DAs dónde registrar los servicios.

Las clases principales que forman el paquete XSLP son las encargadas de realizar la función de cliente y servidor. Estas clases son: `XSLPClient` y `XSLPServer`. La clase `XSLPClient` se encarga de implementar los método que permiten el envío de las peticiones propias del protocolo. Por su parte, la clase `XSLPServer` es la encargada de tratar los mensajes XSLP recibidos, crear y enviar la respuesta correspondiente, tanto si se trata de un asentimiento como de la notificación de cualquier posible error que se haya producido.

Podemos observar la complejidad existente en la arquitectura en los distintos diagramas de clases del código proporcionado. Este ha sido el punto de partida para el diseño de XSRP, integrándolo con la arquitectura existente.

5.2. Casos de Uso XSRP

En la figura 5.4 podemos ver el diagrama de casos de uso, donde se representan los casos de uso correspondientes, en este caso, todas las operaciones del protocolo XSRP:

- Inicialización del SA
- Inicialización del DA
- Registrar un Servicio (Register Service)
- Actualizar un Servicio (Deregister Service)
- Borrar un Servicio (Update Service)

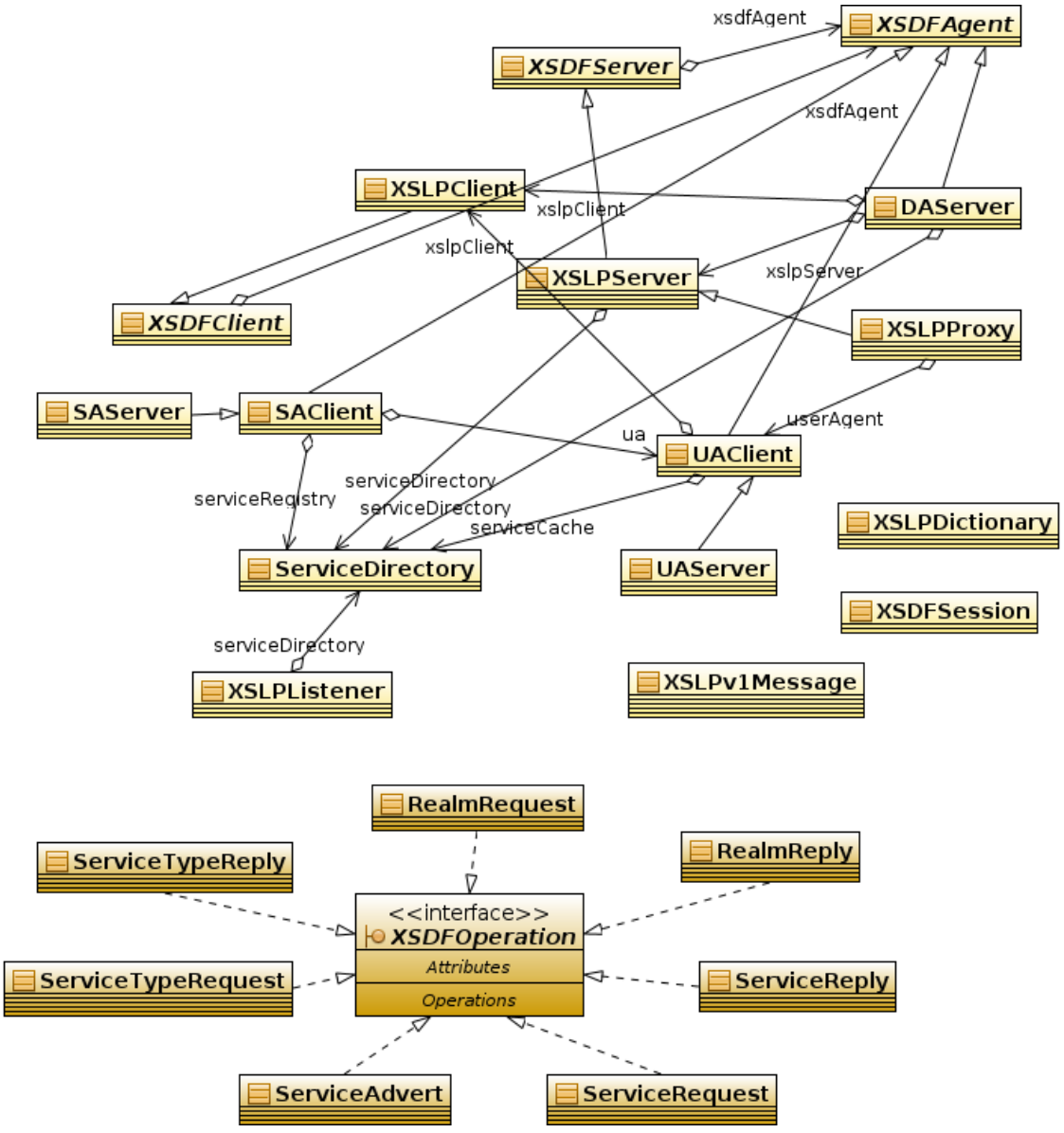


Figura 5.3: Diagrama de clases de XSLP

En las tablas que se muestran a continuación se detallarán de manera esquemática los distintos casos de uso de cada una de las operaciones del protocolo.

De este modo se pretende mostrar la secuencia de interacciones entre el sistema y sus actores en respuesta a un evento que inicia el actor principal sobre el propio sistema.

Para todos los casos el escenario principal es el que se encuentra libre de errores y realiza la operación con éxito. En los escenarios alternativos podemos apreciar las restantes posibilidades que pueden tener lugar cuando se realiza la operación y que dan lugar a diferentes errores.

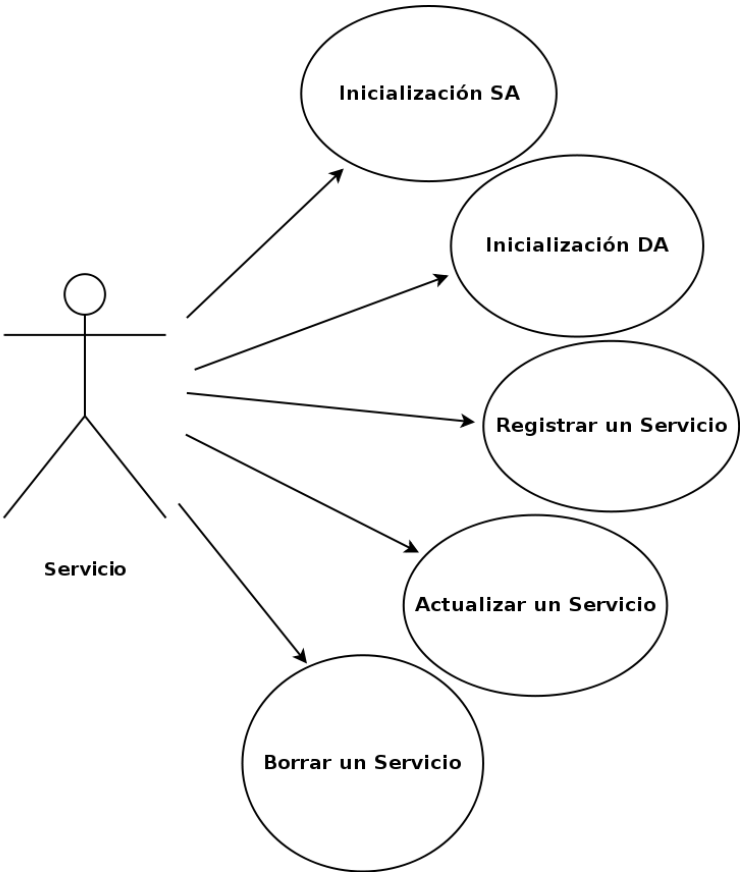


Figura 5.4: Diagrama de Casos de Uso

Nombre	Inicialización del SA
Actores	Servicio
Objetivo	Inicializar el SA
Precondiciones	—
Postcondiciones	SA Inicializado
Escenario Básico	1. Pre-configuración del SA (fichero de configuración) 2. Búsqueda de DA con XSLP
Escenario alternativo	1a. No hay respues del DA. 1. Se reintenta con otro DA o se pasa a *Stand-Alone*.

Nombre	Inicialización del DA
Actores	Servicio
Objetivo	Inicializar el DA
Precondiciones	—
Postcondiciones	DA Inicializado
Escenario Básico	1. Pre-configuración del DA (fichero de configuración) 2. El DA envía un *DAAdvertisement*

Nombre	Registrar Servicio
Actores	Servicio
Objetivo	Registrar un Servicio en el DA
Precondiciones	Inicialización de los Agentes
Postcondiciones	Servicio registrado en el DA
Escenario Básico	1. La aplicación solicita registrar un servicio. 2. El SA crea el mensaje con la operación "RegisterService" y lo codifica con XBE32. 3. El SA envía el mensaje al DA. 4. El DA recibe el mensaje y lo decodifica. 5. Obtener la lista de los *Scopes* donde se va a registrar el servicio. 6. Comprobar que no hay otro servicio registrado con el mismo Id en el *Scope*. 7. Comprobar que no hay otro servicio del mismo tipo registrado con una política de selección incompatible. 8. El DA registra el servicio. 9. El DA elige un intervalo de actualización y crea un "registerServiceAck". 10. El DA envía el mensaje al SA. 11. El SA recibe el mensaje y lo decodifica. 12. El SA comprueba el ACK.
Escenario alternativo	1a. Los *Scopes* no han sido listados en la cabecera. 1. Mensaje abortado.
Escenario alternativo	2a. Otro servicio registrado con el mismo Id. 1. Se crea un error SERVICE_COLLISION.
Escenario alternativo	3a. Existe un servicio del mismo tipo registrado con política de selección incompatible. 1. Se crea un error INCOMPATIBLE_POLICY.
Escenario alternativo	4a. No hay respuesta del DA 1. Se reintenta con otro DA o se pasa al modo *Stand-Alone*

Nombre	Actualizar Servicio
Actores	Servicio
Objetivo	Actualizar un Servicio en el DA
Precondiciones	Inicialización de los Agentes
Postcondiciones	Servicio actualizado en el DA
Escenario Básico	1. La aplicación solicita actualizar un servicio. 2. El SA crea el mensaje con la operación "UpdateService" y lo condifica con XBE32. 3. El SA envía el mensaje al DA. 4. El DA recibe el mensaje y lo decodifica. 5. Obtener la lista de los *Scopes* donde se va a registrar el servicio. 6. Comprobar que existe el registro del servicio que va a ser actualizado. 7. Si la petición del actualización contiene un elemento Selection Information, comprobar que ha sido registrado con la misma política de selección. 8. El DA actualiza el servicio en todos los *Scopes* en los que está registrado. 9. El DA elige un invervalo de actualización y enviar un "updateServiceAck".
Escenario alternativo	1a. Los Scopes no han sido listado en la cabecera. 1. Mensaje abortado
Escenario alternativo	2a. No hay un servicio registrado con el Id del servicio a actualizar. 1. Se crea un error SERVICE_NOT_FOUND.
Escenario alternativo	3a. La petición contiene un elemento *Selection Information* con una distinta política de selección que el servicio registrado. 1. Se crea un error INCOMPATIBLE_POLICY.
Escenario alternativo	4a. No hay respuesta del DA 1. Se reintenta con otro DA o se pasa al modo *Stand-Alone*

Nombre	Borrar Servicio
Actores	Servicio
Objetivo	Borrar un Servicio en el DA
Precondiciones	Inicialización de los Agentes
Postcondiciones	Servicio borrado en el DA
Escenario Básico	1. La aplicación solicita borrar un servicio. 2. El SA crea el mensaje con la operación "DeregisterService" y lo condifica con XBE32. 3. El SA envía el mensaje al DA. 4. El DA recibe el mensaje y lo decodifica. 5. Obtener la lista de los *Scopes* donde se va a registrar el servicio. 6. Comprobar que existe el registro del servicio que va a ser borrado. 7. El DA borra el servicio especificado en todos los *Scopes* en los que había sido registrado. 8. El DA crea un "DeregisterServiceAck". 9. El DA envía el mensaje al SA. 10. El SA recibe el mensaje y lo decodifica. 11. El SA comprueba el ACK.
Escenario alternativo	1a. Los Scopes no han sido listados en la cabecera. 1. Mensaje abortado.
Escenario alternativo	2a. No hay un servicio registrado con el Id del servicio que se quiere borrar. 1. Se crea un error SERVICE_NOT_FOUND.
Escenario alternativo	4a. No hay respuesta del DA 1. Se reintenta con otro DA o se pasa al modo *Stand-Alone*

Una vez descritos los distintos casos de uso de XSRP, mostrando tanto los escenarios básicos como los alternativos, que tendrán lugar en caso de producirse algún tipo de error, se muestra el API del Agente de Servicio (`serviceAgent`).

En el cuadro que se muestra a continuación se pueden ver todos los métodos que ofrece, así como los parámetro que recibe y devuelve. Este API es el que se ofrece a la aplicación para realizar las operaciones básicas del protocolo. Esta clase se encargará de realizar los procedimientos pertinentes para llevar a cabo las operaciones.

API *serviceAgent*	
Realm	getRealm()
XSDFConfiguration	getXSDFConfiguration()
void	deregisterService(Real realm, Service srv)
void	deregisterService(Service srv)
void	registerService(Realm realm, Service srv, int life)
void	registerService(Service srv, int life)
void	updateService(Realm realm, Service srv)
void	updateService(Service srv)

La aplicación dispondrá de estos métodos para la realización de las distintas operaciones. Esta clase se encargará de llamar a los métodos correspondientes de la parte cliente del Agente de Servicio que se corresponde con la clase `SAClient`. Los métodos de esta clase, se encargan de formar los mensajes correspondientes a cada una de las operaciones y mandarlos al servidor encargado de tratar la petición y enviar la respuesta adecuada.

Para cada operación, este API nos proporciona dos métodos, los cuales reciben distinto número de parámetros en función de si se proporciona o no el `realm` donde se realiza la operación. En caso de que no se proporcione este parámetro, se obtendrá mediante la llamada al `getRealm()` de esta misma clase, que se puede ver en la tabla anterior.

Los métodos `registerService(Service srv, int life)` y `registerService (Realm realm, Service srv, int life)` se encargan de crear el mensaje XSRP con la operación "RegisterService" para que sea registrado en el DA previamente descubierto mediante XSLP. En caso de que el parámetro `life` sea menor que cero se procederá a realizar un proceso de actualización periódica en intervalos delimitados por el valor absoluto de dicho parámetro. Este

parámetro será opcional y en caso de no proporcionarse se asignará un valor por defecto. Los métodos `updateService(Service srv)` y `updateService (Realm realm, Service srv)` se encargarían de crear el mensaje XSRP con la operación "updateService" para actualizar un servicio previamente registrado en el DA. Por último, los métodos `deregisterService (Service srv)` y `deregisterService(Realm realm, Service srv)` son los encargados de crear un mensaje XSRP con la operación "DeregisterService" para que se borre del DA.

Capítulo 6

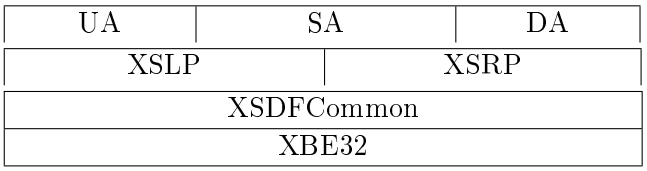
Implementación de un Cliente-Servidor XSRP

En este capítulo se pretende explicar la implementación los distintos agentes que componen el protocolo XSRP (cliente y servidor), así como las operaciones que realiza (Registrar, Actualizar y Borrar un servicio).

Para facilitar la comprensión de las distintas operaciones, se presentarán en las siguientes secciones de este capítulo los diagramas de flujo de todas las operaciones, donde se puede apreciar de manera sencilla los pasos a seguir para llevar a cabo cada uno de los procesos.

Para mostrar en detalle la implementación de los Casos de Uso más significativos del código desarrollado, se presentan a continuación los diagramas de secuencias de los métodos que realizan las operaciones básicas del protocolo. Se ha decidido realizar este tipo de diagramas por ser los más efectivos para modelar la interacción entre objetos de un sistema. Los diagramas de secuencia nos mostrarán la interacción del conjunto de objetos involucrados en la operación a través del tiempo.

En primer lugar se muestra la arquitectura conjunta de los protocolos XSLP y XSRP.



Se puede ver que ambos protocolos, XSLP y XSRP, comparten las dos capas inferiores de la arquitectura sobre la que se han implementado, `XBE32` y `XSDFCommon`, encargadas de la codificación y decodificación y el transporte de los mensajes respectivamente. También se aprecia cómo los agentes involucrados en XSLP son el agente de usuario (UA) y el agente de servicio (SA), en este caso el primero actuaría como cliente, mientras que el segundo actuaría como servidor. En XSRP los agentes involucrados son el agente de servicio, que en este caso actúa como cliente, y el agente de directorio, que ahora es el que cumple la función de servidor.

6.1. Cliente XSRP

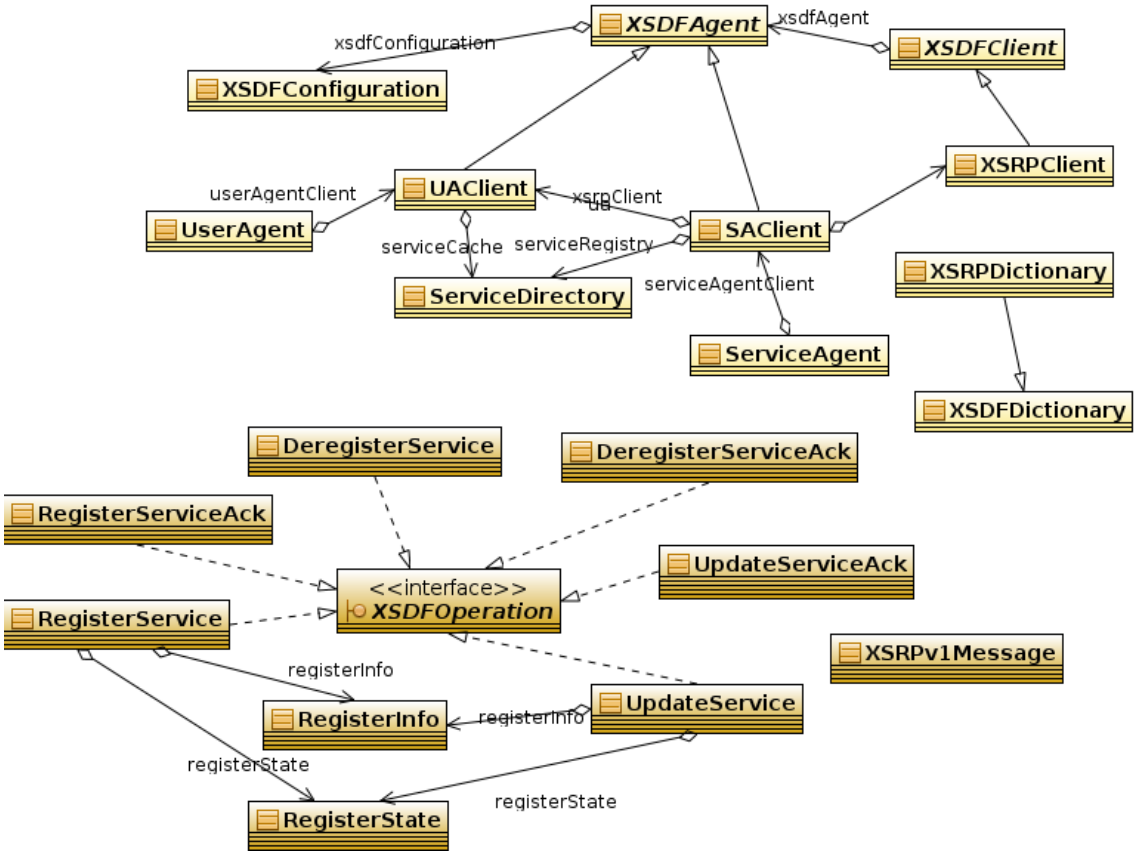


Figura 6.1: Diagrama de clases del cliente XSRP

El cliente XSRP tiene un método que se encarga del intercambio de mensajes XSRP. Este método recibe un objeto `XSDFSession` y un mensaje XSRP (objeto `XSRPv1Message`), y se encarga de mandar dicho mensaje y esperar

la respuesta que se recibe como respuesta a éste. `XSDFSession` implementa los métodos de envío y recepión de mensajes XSDF. Para crear el objeto `XSDFSession` se utiliza el método `createXSRPSession()` que recibe como parámetros el *realm* al que pertenece, el protocolo, en este caso XSRP, el orden de preferencia de los protocolos de transpote (UDP y TCP) y una lista con los DAs con los que se realiza la comunicación para la realización de las distintas operaciones.

Para cada una de las operaciones del protocolo (registrar, actualizar y borrar) existe un método que recibe un objeto `XSRPSession` y un objeto de tipo *Service*. Con esto se tiene toda la información necesaria para el envío de la petición correspondiente a cada operación.

Antes de llamar al método `exchageXSRPMessage()`, encargado del intercambio de mensajes, se forma el mensaje XSRP correspondiente, con todos los campos necesarios para ser enviado, su cabecera bien formada y en el campo donde se indica la operación que se realiza con ese mensaje, se crea y añade al mensaje la operación correspondiente. Una vez formado el mensaje se llama al método anteriormente comentado, que se encarga de enviar la petición y esperar a recibir la respuesta correspondiente, ya sea un asentimiento o un mensaje de error consecuencia de algún fallo en la realización de la operación en cuestión.

En la figura 6.2, se muestra un diagrama de secuencia donde se expone de manera más detallada el método `exchangeXSRPMessage()` encargado del intercambio de mensajes del protocolo XSRP haciendo uso de la arquitectura ya comentada. No se detallan absolutamente todos los métodos involucrados, pero sí los considerados más importantes para comprender la ejecución y la arquitectura empleada.

El método `exchangeXSRPMessage()` es el que aparece en los diagramas de secuencia correspondientes a las distintas operaciones. En estos diagramas, no se detallarán los pasos sí descritos en este último para una mayor claridad y legibilidad.

6.2. Servidor XSRP

La principal función que realiza el servidor de XSRP es gestionar los mensajes XSRP que son recibidos. Para ello, en primer lugar se comprueba el tipo de operación que contiene el mensaje, la cual puede ser `RegisterService`

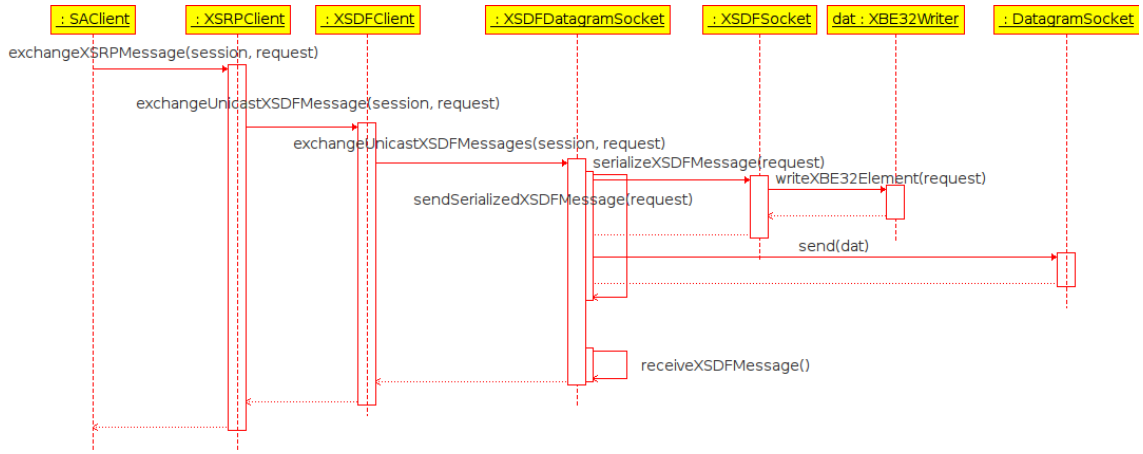


Figura 6.2: Diagrama de Secuencias: exchangeXSRPMessage

para registrar un servicio, `UpdateService` para actualizar un servicio previamente registrado o `DeregisterService` para borrar un servicio previamente registrado.

La clase `XSRPServer` se encarga de procesar los mensajes XSRP recibidos comprobando si son correctos sintácticamente. Sin embargo, el procesamiento en sí de las operaciones se realiza en la clase `DAServer`. Para ello el Agente de Directorio se subscribe al XSRPServer, de modo que cada vez que se recibe una operación XSRP, el Agente de Directorio la procesa y devuelve el resultado para que el `XSRPServer` genere la operación de respuesta correspondiente. De este modo, es posible independizar el Agente de Directorio de los mensajes en sí, lo que permite que, por ejemplo, un mensaje pueda contener varias operaciones simultáneamente.

Para cada tipo de operación, se realiza un procesado que finaliza con la realización de la operación correspondiente y el tratamiento de errores pertinente en cada uno de los casos. Para ello, entra en escena la clase `ServiceDirectory` donde se almacenan todos los servicios y que permite añadir, actualizar y borrar un determinado servicio mediante la llamada a los métodos `put()`, `update()` y `remove()` respectivamente, además de realizar búsquedas por tipo de servicio.

6.3. Cliente del Agente de Servicio

Para la realización de este proyecto se han implementado las clases necesarias que correspondería a los agentes que tendrían que intervenir en el re-

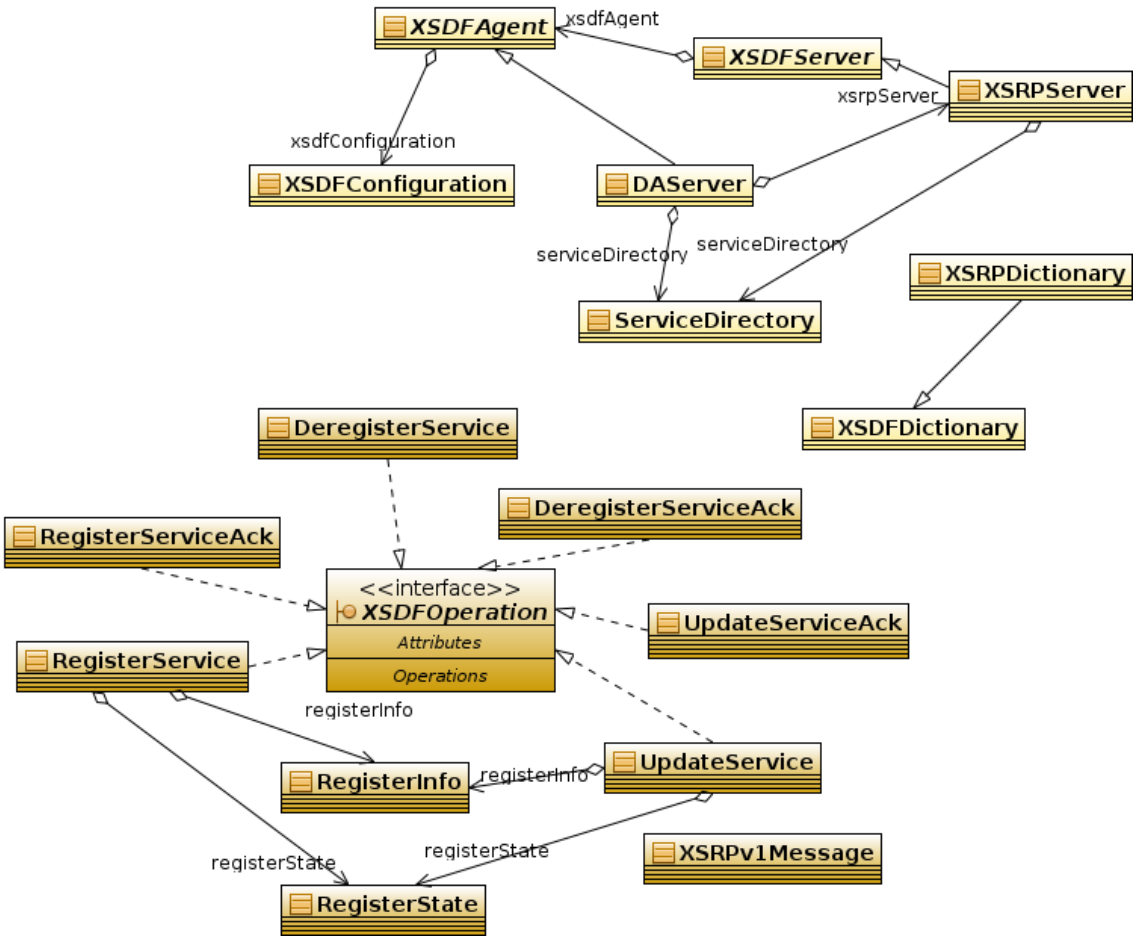


Figura 6.3: Diagrama de clases del servidor XSRP

gistro de un servicio utilizando XSRP. Del mismo modo, también se incluye XSLP para poder llevar los procedimientos de descubrimiento de servicio necesarios para el correcto funcionamiento del protocolo. Por ejemplo, XSLP se emplea para encontrar los DAs dónde se registran los servicios.

En la figura 6.4 se puede apreciar la arquitectura de clases que conforman la parte del cliente del agente de servicio. Como se puede ver en dicha imagen, el cliente del agente de servicio tiene un cliente XSRP, el cual es el encargado de enviar las peticiones de las distintas operaciones que se encarga de realizar el protocolo.

En esta clase se implementa el método `createXSRPSession` que como se ha comentado anteriormente devuelve un objeto `XSDFSession` que encapsula los

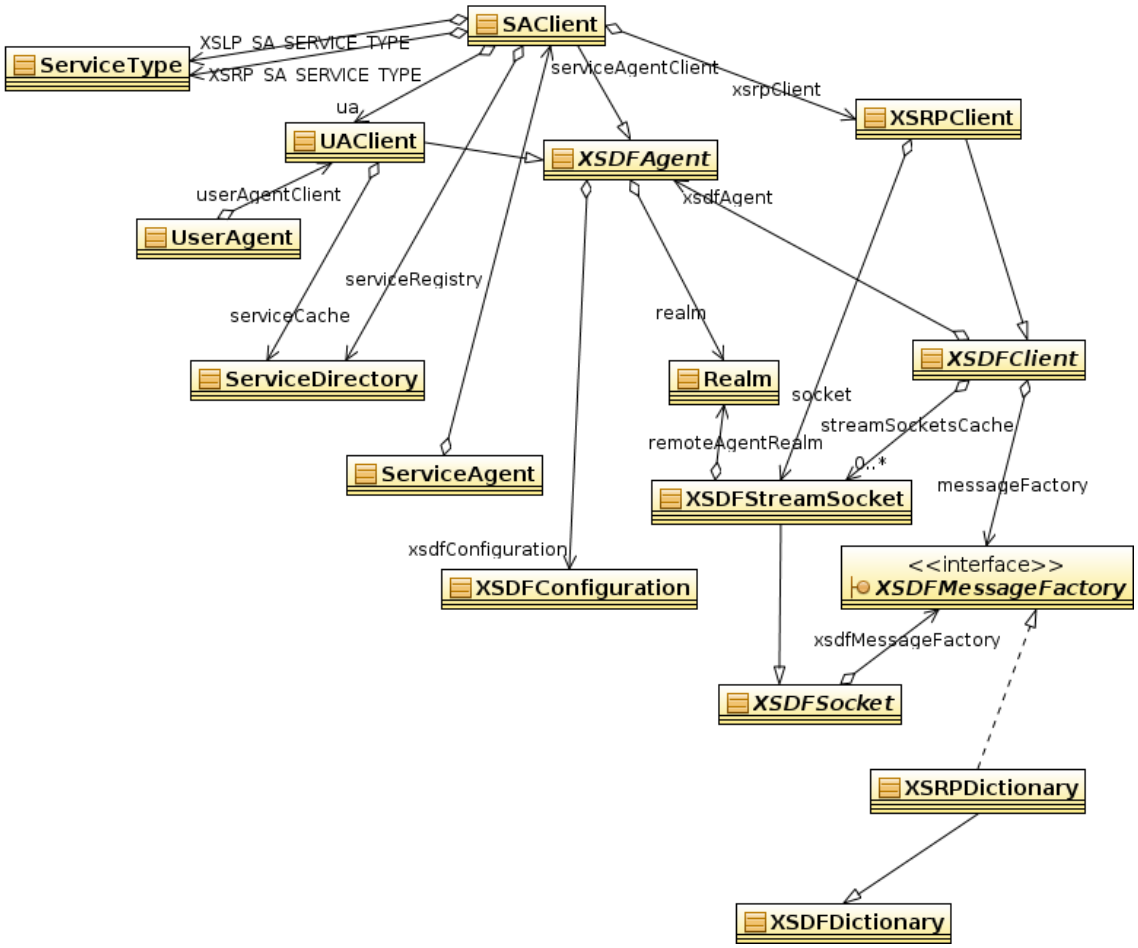


Figura 6.4: Diagrama de clases del *SAClient*

DAs, encontrados mediante XSLP gracias al método `getDirectoryAgents()`, para la realización de las operaciones del protocolo y el orden de preferencia de los protocolos de transporte.

Además de este método, en esta clase se implementa un método por cada operación del protocolo encargados de crear el objeto XSDFSession y la llamada al método pertinente de la clase XSRPClient encargado de crear el mensaje para ser enviado para realizar la operación en cuestión.

6.4. Servidor del Agente de Directorio

Al igual que se ha comentado en la sección anterior con el cliente del agente de servicio, en este caso se trata de de la parte del servidor, la cual

concierne en el caso que nos compete a la parte del servidor del agente de directorio.

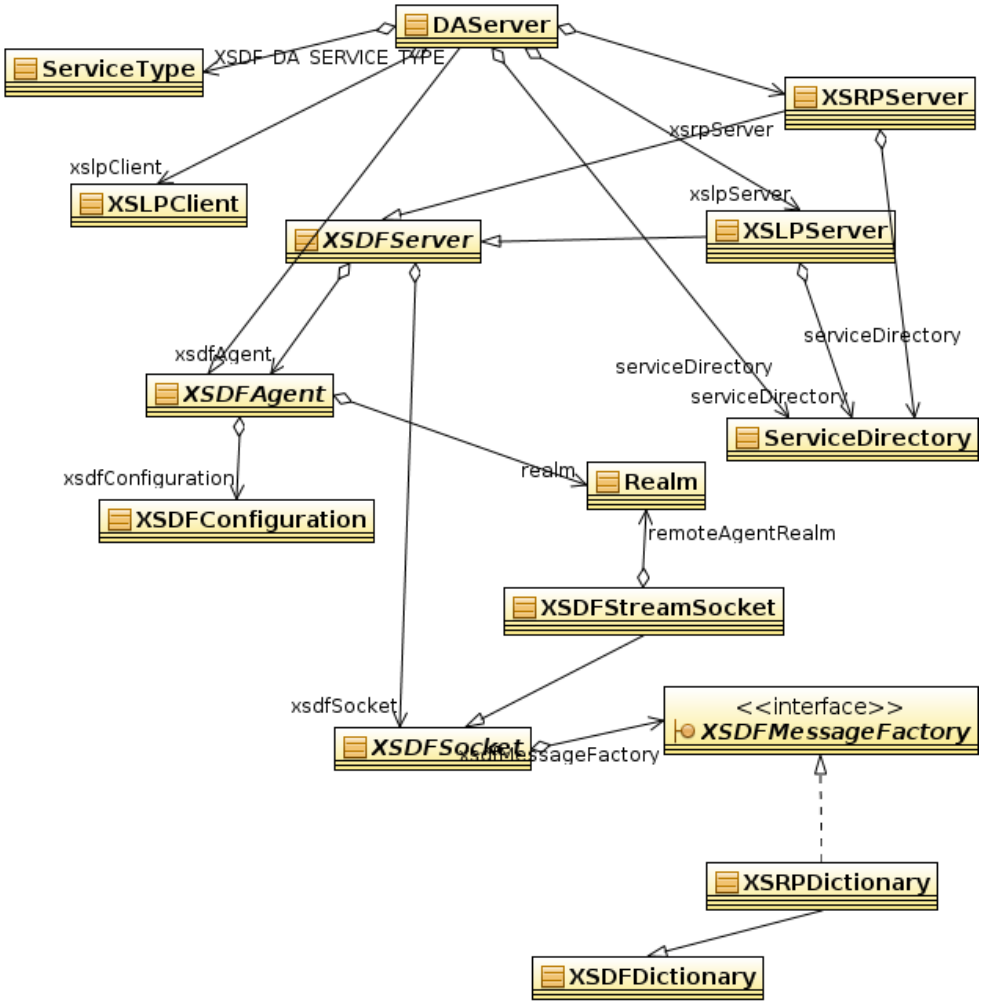


Figura 6.5: Diagrama de clases del *DAServer*

En la figura 6.5 se puede ver el diagrama de clases del servidor del agente de directorio, donde podemos observar que en dicho servidor coexiste un servidor XSRP y un servidor XSLP ya que para ambos protocolos es necesario responder a peticiones que llegan al agente de directorio. En el caso que nos concierne, el servidor del agente de directorio será el encargado de responder a las peticiones correspondientes a las distintas peticiones realizadas por el protocolo XSRP.

En esta clase existe el método `createDAService` que se encarga de crear un servicio que se corresponde con el DA en cuestión, al cual se llama en el contructor de la clase para que cuando se realicen las consultas XSLP para buscar los DAs presentes en el *realm* éste pueda ser descubierto.

Cabe destacar, como se ha comentado en el diseño, el funcionamiento asíncrono del Agente de Directorio, el cual se registra a los eventos del servidor XSRP. De este modo, cada vez que recibe un mensaje XSRP se procede a su tratamiento, creando la respuesta correspodiente.

Para que sea más sencillo entender como realiza su función el servidor del agente de directorio, en la figura 6.6 se muestra un diagrama de secuencia donde se detallan los pasos que sigue el servidor para realizar su función. Esto consiste en recibir los mensajes, procesarlos, crear la respuesta adecuada y enviarla al cliente.

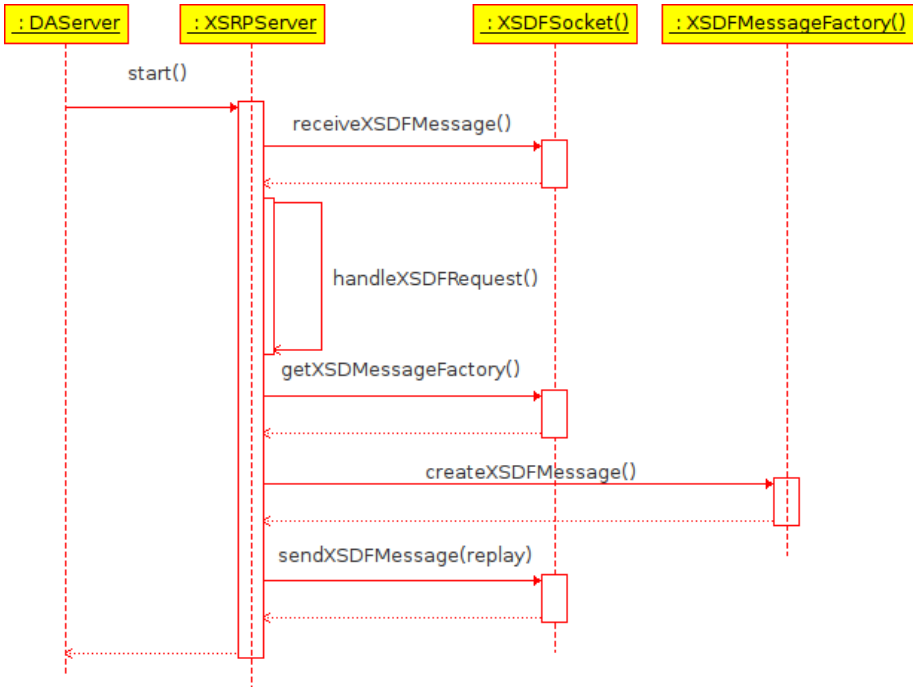


Figura 6.6: Diagrama de Secuencias: DAServer

Para ello se llevan a cabo los siguientes pasos. El `DAserver` llama al método `start()` del `XSRPServer`. Esta clase hereda de la clase `XSDFServer`, la cual

es `Thread`. Dentro de ese método `run()`, en un bucle infinito, se ejecuta el método `receiveXSDFMessage()` encargado de la recepción de los mensajes. Una vez recibido el mensaje se realiza su procesamiento mediante la ejecución del método `handleXSDFRequest()`. Este método identifica la operación del mensaje recibido (registerService, deregisterService, updateService), crea el mensaje de respuesta correspondiente y lo envía al cliente.

6.5. Register Service

Esta operación, definida en el capítulo anterior, se ha implementado siguiendo el Diagrama de Secuencias mostrado en la figura 6.8. Los pasos para la realización de esta operación se pueden observar en el diagrama de flujo de la figura 6.7.

El cliente, en este caso la aplicación que ofrece el servicio, ejecuta el método registerService() de la clase `ServiceAgent`. En el cuerpo de este método se llama al método `registerService()` del cliente del Agente de Servicio. Dentro de ese método se crea un objeto de la clase `XSDFSession` cuya función ya ha sido comentada.

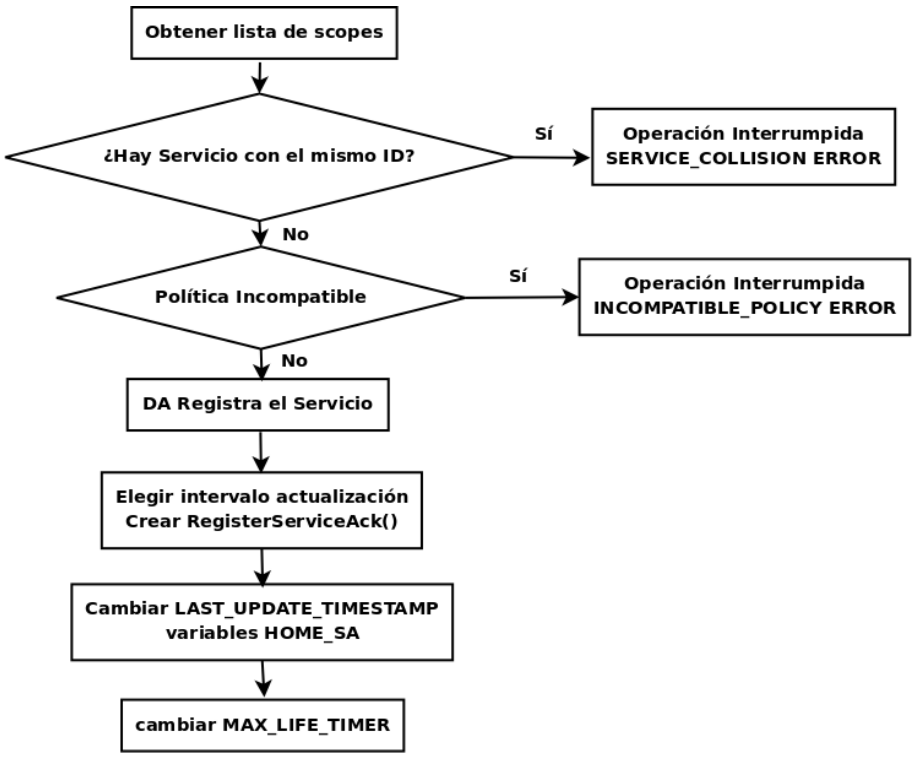


Figura 6.7: Pasos a seguir para **Registar un Servicio**

Una vez hecho esto, se instancia un objeto de la clase XSRPClient y se llama al método `registerService2()`. Este método se encarga de crear el mensaje (instancia de la clase `XSRPv1Message`) que se enviará al DA para registrar el servicio. Para rellenar los campos del mensaje es necesario crear una instancia de la clase `RegisterService` y añadirlo al atributo `Operations` de la clase `XSRPv1Message`, de este modo, cuando el servidor reciba el mensaje, sabrá la operación que debe realizar y el mensaje de respuesta que debe mandar.

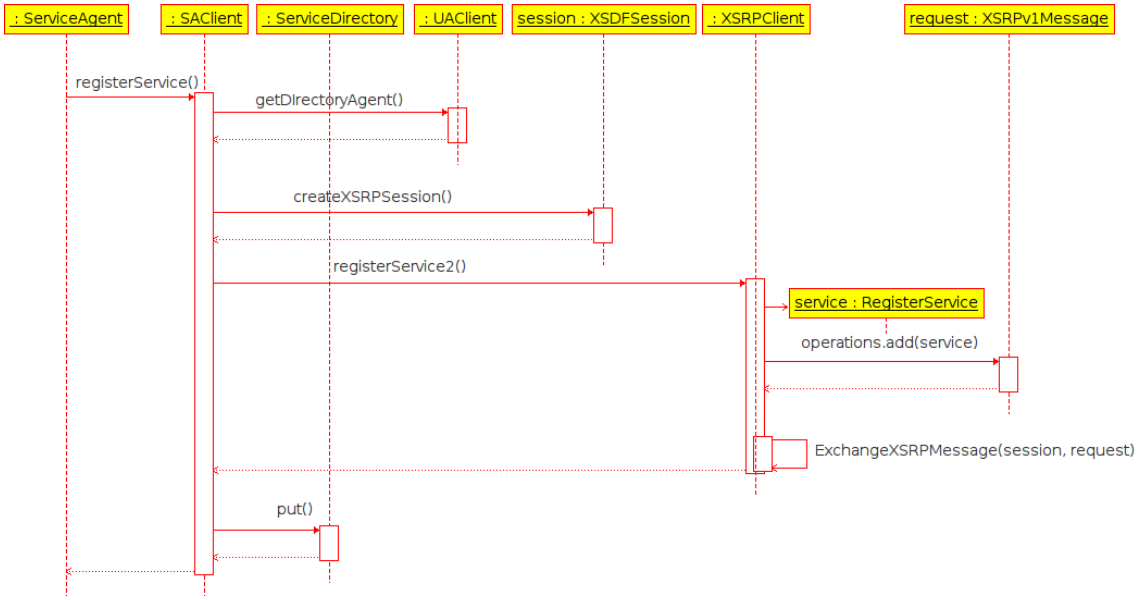


Figura 6.8: Diagrama de Secuencias: Register Service

El siguiente paso es llamar al método `exchangeXSRPMessage()`, donde se le pasan como parámetros el `XSRPv1Message` creado anteriormente y el objeto `XSDFSession` donde están encapsulados los DAs en los que se debe registrar el servicio. Este método, llama a otros métodos más genéricos de la arquitectura XSDF que se encargan de enviar y esperar la respuesta del mensaje, `exchangeXSRPMessage()` emplea temporizadores de modo que si no recibe respuesta por algún error, retransmite el mensaje.

Cuando el servidor, en este caso el DA, recibe un mensaje, comprueba que éste pertenece al protocolo XSRP. En este caso la clase `XSRPServer` es la encargada de realizar el tratamiento adecuando del mensaje recibido

y, dependiendo de lo que se incluya en el atributo operations del mensaje realiza la operación indicada y crea un mensaje de respuesta.

En este caso, que la operación del mensaje XSRP recibido es `RegisterService`, en primer lugar se comprueba si ha sido registrado previamente otro servicio con el mismo id. En este caso se crea una operación de error que es lo que devuelve el método encargado de gestionar los mensajes recibidos.

En caso de que este error no se haya producido, se comprueba si la política de selección es compatible con los servicio del mismo tipo que ya hayan sido registrados. Si no se trata de una política compatible, se crea una operación de error, del mismo modo que en el caso anterior, pero con el error correspondiente.

Si, por el contrario, no ha ocurrido ningún problema, se procede a registrar el servicio y se crea una respuesta `RegisterServiceAck` para confirmar que la operación se ha realizado con éxito.

El servicio también es registrado en el SA mediante la llamada al método `put()` de la clase `ServiceDirectory`. Esto se realiza siempre ya que en caso de que el DA falle, el servicio debe ser anunciado por el SA de todas formas.

6.6. Deregister Service

En este caso, el proceso es análogo al anterior. En la figura 6.9 podemos ver el diagrama de flujo que muestra los pasos necesarios para borrar un servicio previamente registrado. En el diagrama de secuencias de la figura 6.10 podemos ver como los pasos son los mismos y los métodos que se utilizan para la realización de la operación son complementarios.

En este caso la operación que se añade en el mensaje XSRP es una instancia de la clase `DeregisterService`, el método al que llama el servidor para realizar la operación es un `remove()` en la clase que actúa como repositorio central, `ServiceDirectory`, en lugar del `put()` del caso anterior, y el mensaje que enviará el servidor una vez compruebe el tipo de operación del mensaje recibido será un `DeregisterServiceACK()` si todo ha ocurrido según lo previsto, o se creará el mensaje de error correspondiente en el caso de que algo no haya ocurrido correctamente.

Del mismo modo que en la operación de registro, el servicio también se borra del SA mediante la llamada al método remove() de la clase `ServiceDirectory` ya que el SA también debe dejar de ser anunciado también por el DA.

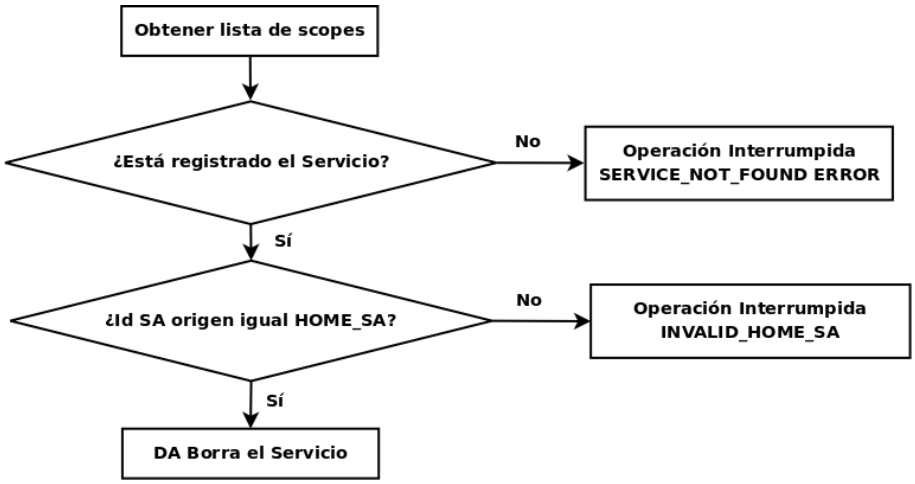


Figura 6.9: Pasos a seguir para **Borrar un Servicio**

6.7. Update Service

Por ultimo, para la realización de esta operación se ejecuta el método `updateService()`, y los pasos a seguir son equivalentes a los descritos anteriormente para registrar y borrar un servicio previamente registrado.

En la figura 6.11 se pueden ver los pasos a seguir para actualizar un servicio previamente registrado.

En la figura 6.12 podemos ver el diagrama de secuencias del método encargado de actualizar el servicio, análogo a los anteriores

En este caso la operación que se añade en el mensaje XSRP es una instancia de la clase `UpdateService`, el método al que llama el servidor para realizar la operación es un `update()` en la clase que actúa como repositorio central, `ServiceDirectory`, en lugar del `put()` o `remove()` de los casos anteriores, y el mensaje que enviará el servidor una vez compruebe el tipo de operación del mensaje recibido será un `UpdateServiceACK()` si no ha ocurrido ningún error, o se creará un mensaje de error que informará de lo sucedido en el caso de que algo no haya ido bien.

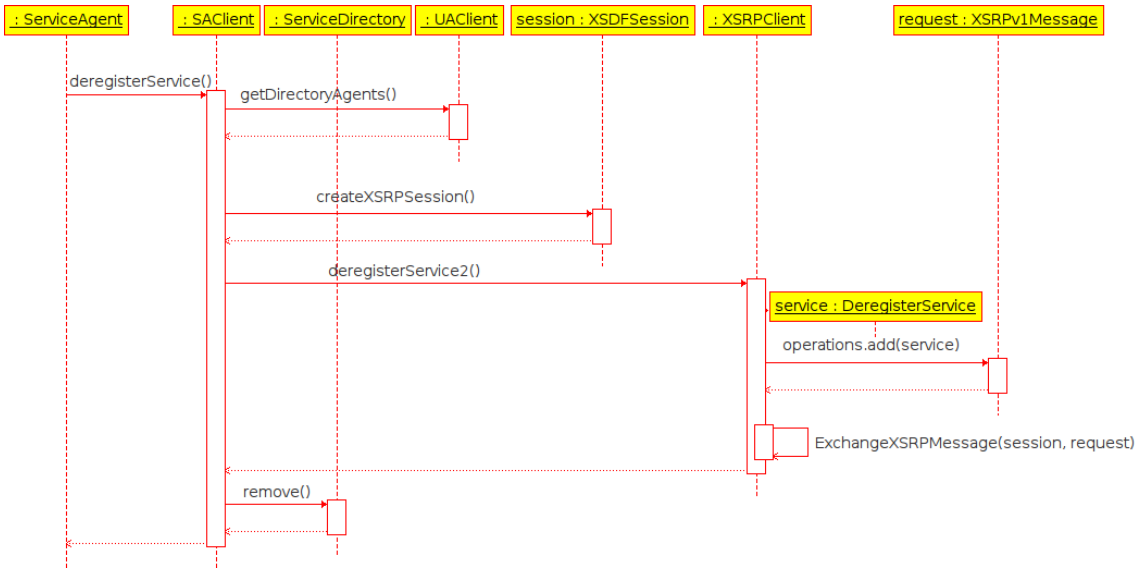


Figura 6.10: Diagrama de Secuencias: Deregister Service

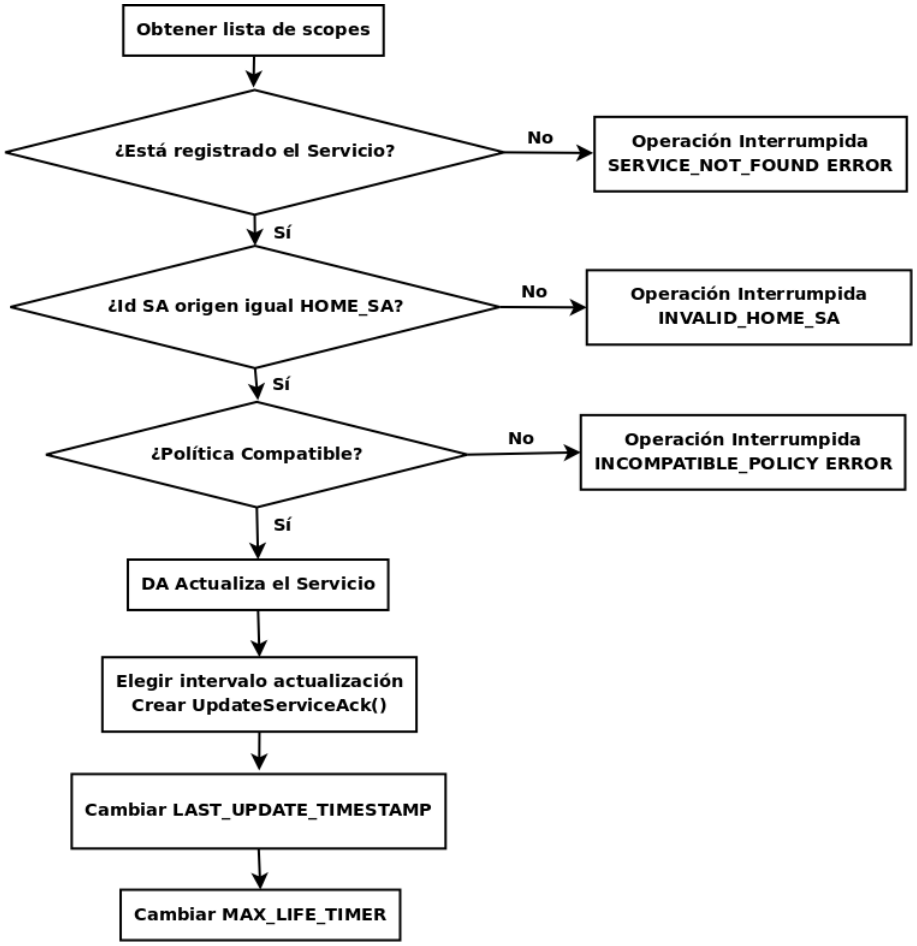


Figura 6.11: Pasos a seguir para **Actualizar un Servicio**

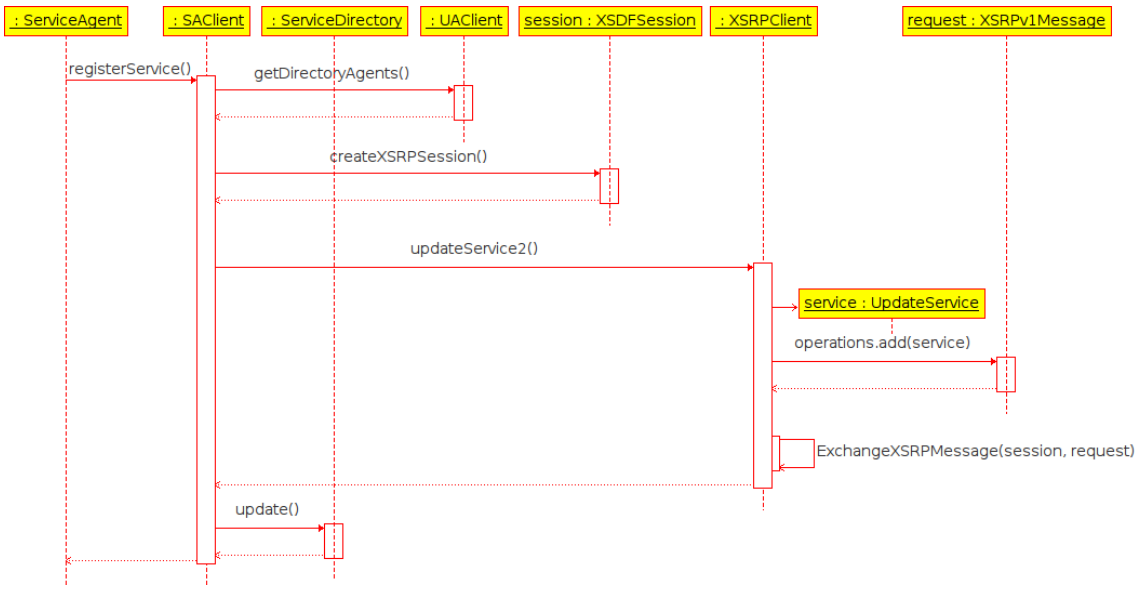


Figura 6.12: Diagrama de Secuencias: Update Service

Capítulo 7

Fase de Pruebas

En el ámbito de este Proyecto Fin de Carrera la fase de pruebas tiene una importancia vital debido a que se trata del diseño e implementación de una librería cliente-servidor de la cual se presupone un uso intensivo y, por tanto, debe aportar unos valores de fiabilidad que le hagan factible para ser utilizada por cualquier tipo de aplicación, asegurando un buen funcionamiento de la misma.

Debido a que la fiabilidad es un factor determinante, se han diseñado una serie de pruebas para comprobar el funcionamiento del protocolo tanto en casos normales como frente a la presencia de errores. Sin embargo, es imposible tener una cobertura absoluta, en la cual se comprueban todas las posibles entradas que un usuario final pueda hacer, debido a la posibilidad de trabajar con parámetros de entrada que pueden ser arbitrarios, con lo cual el número de posibilidades sería prácticamente ilimitado. Debido a eso se ha optado por la decisión de realizar una muestra representativa de los casos más habituales tanto de éxito como de error mediante la técnica de pruebas de caja negra, donde se especificarán tanto los parámetros de entrada como los mensajes de petición y respuesta intercambiados por los agentes y el resultado de la prueba.

Para tal fin, el objetivo de esta sección será desarrollar un conjunto intensivo de casos de prueba que nos permitan asegurar esta fiabilidad en cada una de las operaciones que realiza esta librería, haciendo hincapié en los errores más plausibles, pero sin descuidar cualquier tipo de error factible.

Debido a esto, se ha tomado la decisión para el diseño del conjunto de casos de prueba de basarse en los resultados del análisis inicial de las funcionalidades a través de los caso de uso, comentados en el capítulo cinco.

Las motivaciones que han derivado en esa decisión es la importancia de que las pruebas sean definidas en el momento del análisis con el fin de que los requisitos de esta librería sean verificados de una manera rigurosa y no haya lugar a duda de que se ha logrado la funcionalidad prevista.

Para la realización de las pruebas se pasará como parámetro de entrada un fichero .xml con la información del servicio que se pretende registrar, actualizar o borrar. A no ser que se diga lo contrario, el fichero de servicio que se utilizará será el siguiente:

```
<service>
 <id> aaaaaaaa-aaaa-4aaa-8aaa-aaaaaaaaaaaa
     (0xAAAAAAAAAAAA4AAA8AAAAAAAAAAAAAA)
 </id>
 <serviceState>
   <stateTimestamp> 0 </stateTimestamp>
   <selectState>
     <workload> 0 </workload>
   </selectState>
 </serviceState>
 <serviceMainInfo>
   <serviceType>
     <type> printer </type>
   </serviceType>
   <alias> Old printer </alias>
   <selectInfo>
     <policies> Max Priority (0x0003),
                Least Used (0x0004)
     </policies>
     <priority> 5 </priority>
   </selectInfo>
   <serviceOptions>
     <color  ext="true" c="true" e="false"
     type="boolean"> false
     </color>
     <duplex ext="true" c="true" e="false"
     type="boolean">  false
     </duplex>
   </serviceOptions>
 </serviceMainInfo>
 <serviceNetInfo>
   <inet>
     <ipv4Addrs> 169.254.10.10 (0xA9FE0A0A) </ipv4Addrs>
     <hostname> printer1.local. </hostname>
```

```
    </inet>
    <protocol>
      <protoName> acme:printdirect </protoName>
      <transPorts> tcp/9200 (0x000623F0) </transPorts>
    </protocol>
    <protocol>
      <protoName> lpr </protoName>
      <transPorts> tcp/515 (0x00060203) </transPorts>
    </protocol>
  </serviceNetInfo>
  <serviceAddInfo>
    <url> http://printer.local/index.html </url>
    <vendor> Acme Corporation </vendor>
    <vendorURL> http://www.acme.com/index.html </vendorURL>
    <model> Acme Matrix Printer 1000 </model>
    <modelURL> http://www.acme.com/printers/mp1000.html
    </modelURL>
    <version> 3.2 </version>
    <location> Basement </location>
  </serviceAddInfo>
</service>
```

Para la realización de las distintas pruebas tenemos una máquina donde se ejecuta la clase `DAServer`, encargada de realizar la función de servidor en el protocolo XSRP. En una máquina distinta, situada en la misma red que la anterior, se ejcuta la clase `SAClient` que recibe como parámetros la operación a realizar (registrar, actualizar o borrar), el fichero .xml anteriormente comentado, o en su defecto otro de similares características y, en caso de tratarse de la operación "registrar" un parámetro opcional "lifetime" que indica el tiempo de vida durante el cual el servicio registrado es válido. En caso de no proporcionarse este último se asignará un valor por defecto.

7.1. Pruebas del caso de uso Register Service

Caso de Prueba 1

- Descripción: Con la realización de esta prueba, se pretende mostrar el buen funcionamiento de la operación `registerService`, la cual permite registrar un servicio, cuando se le pasa como parámetro de entrada un fichero .xml con la información del servicio que se pretende registrar, el cual posee toda la información del mismo.

En este caso, este servicio no había sido registrado previamente, de modo que no habrá otro servicio con el mismo id que pueda provocar una colisión. La política empleada es compatible con el resto de servicios del mismo tipo, previamente registrados, por lo que no provocará ningún tipo de incompatibilidad de políticas.

Una vez el servicio es registrado, enviando un mensaje XSRP con la operación `registerService`, el servicio quedará registrado y se enviará un mensaje XSRP con la operación `registerServiceAck` para confirmar la operación y comunicar, de este modo, que se ha realizado correctamente y no se ha producido ningún tipo de error, ni de conflicto.

- Mensaje de petición

```
<xsrpv1>
 <header>
   <xid> 0xF1CA2D7B </xid>
   <realm>
     <domain> it.uc3m.es </domain>
     <scope> DEFAULT </scope>
   </realm>
   <source>
     <id> 77740ccb-6a9e-4572-95be-d701c9db104a
          (0x77740CCB6A9E457295BED701C9DB104A)
     </id>
   </source>
   <destination>
     <id> b94ba78f-0f74-44b9-9f12-746d988972bc
          (0xB94BA78F0F7444B99F12746D988972BC)
     </id>
   </destination>
 </header>
 <registerService>
   <realmTarget>
     <scopesMap> true </scopesMap>
   </realmTarget>
   <service>
     <id> aaaaaaaa-aaaa-4aaa-8aaa-aaaaaaaaaaaa
          (0xAAAAAAAAAAAA4AAA8AAAAAAAAAAAAAAA)
     </id>
     <serviceState>
       <stateTimestamp> -982789859 </stateTimestamp>
       <selectState>
```

```
    <resources> 3 </resources>
    <workload> 0.0 </workload>
  </selectState>
</serviceState>
<serviceMainInfo>
  <serviceType>
    <type> printer </type>
  </serviceType>
  <alias> Old printer </alias>
  <selectInfo>
    <policies> Max Priority (0x0003),
               Least Used (0x0004)
    </policies>
    <priority> 5 </priority>
    <weight> 1.0 </weight>
  </selectInfo>
  <serviceOptions>
    <color ext="true" c="true"
           type="boolean"> false
    </color>
    <duplex ext="true" c="true"
            type="boolean"> false
    </duplex>
  </serviceOptions>
</serviceMainInfo>
<serviceNetInfo>
  <inet>
    <ipv4Addrs> 169.254.10.10 (0xA9FE0A0A) </ipv4Addrs>
    <hostname> printer1.local. </hostname>
  </inet>
  <protocol>
    <location> acme:printdirect </location>
    <transPorts> tcp/9200 (0x000623F0) </transPorts>
  </protocol>
  <protocol>
    <location> lpr </location>
    <transPorts> tcp/515 (0x00060203) </transPorts>
  </protocol>
</serviceNetInfo>
<serviceAddInfo>
  <url> http://printer.local/index.html </url>
  <vendor> Acme Corporation </vendor>
  <vendorURL> http://www.acme.com/index.html </vendorURL>
  <model> Acme Matrix Printer 1000 </model>
```

```
        <modelURL> http://www.acme.com/printers/mp1000.html
        </modelURL>
        <version> 3.2 </version>
        <location> Basement </location>
      </serviceAddInfo>
    </service>
    <registerState>
      <selectState>
        <resources> 1 </resources>
        <workload> 0.0 </workload>
      </selectState>
    </registerState>
    <registerInfo>
      <lifetime> 2 </lifetime>
      <selectInfo>
        <policies> Max Priority (0x0003),
                   Least Used (0x0004)
        </policies>
        <priority> 1 </priority>
        <weight> 1.0 </weight>
      </selectInfo>
    </registerInfo>
    <updateInfo>
      <minLife> 3600 </minLife>
      <maxLife> 36000 </maxLife>
    </updateInfo>
  </registerService>
</xsrpv1>
```

- Mensaje de Respuesta

```
<xsrpv1>
 <header>
   <xid> 0xF1CA2D7B </xid>
   <realm>
     <domain> it.uc3m.es </domain>
     <scope> DEFAULT </scope>
   </realm>
   <source>
     <id> b94ba78f-0f74-44b9-9f12-746d988972bc
          (0xB94BA78F0F7444B99F12746D988972BC)
     </id>
   </source>
   <destination>
```

```
      <id> 77740ccb-6a9e-4572-95be-d701c9db104a
           (0x77740CCB6A9E457295BED701C9DB104A)
      </id>
    </destination>
  </header>
  <registerServiceAck>
    <service>
      <id> aaaaaaaa-aaaa-4aaa-8aaa-aaaaaaaaaaaa
           (0xAAAAAAAAAAAA4AAA8AAAAAAAAAAAAAAA)
      </id>
    </service>
    <updateInfo>
      <minLife> 98276755 </minLife>
      <maxLife> 982767558 </maxLife>
    </updateInfo>
  </registerServiceAck>
</xsrpv1>
```

- Resultado: CORRECTO. Para comprobar que se ha realizado correctamente el registro en el Agente de Directorio (DA) se ha realizado una consulta utilizando el protocolo XSLP donde podemos ver que hay un servicio registrado en el DA.

Caso de Prueba 2

- Descripción: En este caso, se pretende registrar un servicio que tiene un id que coincide con el id de uno de los servicios ya registrados.

  Para la realización de esta prueba, se introduce como parámetro un documento .xml con la información del servicio que se pretende registrar, el cual tiene un id no válido por coincidir con el id de uno de los servicios registrados. Como resultado se debe responder al mensaje XSRP con la operación `registerService`, que pretende registrar dicho servicio, con un mensaje XSRP con una operación de error. Éste es de tipo SERVICE_COLLISION.

- Mensaje de petición

```
<xsrpv1>
 <header>
   <xid> 0xF1CA2D7B </xid>
   <realm>
     <domain> it.uc3m.es </domain>
     <scope> DEFAULT </scope>
```

```
  </realm>
  <source>
    <id> 77740ccb-6a9e-4572-95be-d701c9db104a
         (0x77740CCB6A9E457295BED701C9DB104A)
    </id>
  </source>
  <destination>
    <id> 3f44ce10-29fd-48dd-85aa-c7882f4e1c5e
         (0x3F44CE1029FD48DD85AAC7882F4E1C5E)
    </id>
  </destination>
</header>
<registerService>
  <realmTarget>
    <scopesMap> true </scopesMap>
  </realmTarget>
  <service>
    <id> aaaaaaaa-aaaa-4aaa-8aaa-aaaaaaaaaaaa
         (0xAAAAAAAAAAAA4AAA8AAAAAAAAAAAAAAA)
    </id>
    <serviceState>
      <stateTimestamp> -982789859 </stateTimestamp>
      <selectState>
        <resources> 3 </resources>
        <workload> 0.0 </workload>
      </selectState>
    </serviceState>
    <serviceMainInfo>
      <serviceType>
        <type> printer </type>
      </serviceType>
      <alias> Old printer </alias>
      <selectInfo>
        <policies> Max Priority (0x0003),
                   Least Used (0x0004)
        </policies>
        <priority> 5 </priority>
        <weight> 1.0 </weight>
      </selectInfo>
      <serviceOptions>
        <color ext="true" c="true"
               type="boolean"> false
        </color>
        <duplex ext="true" c="true"
```

```
                type="boolean"> false
          </duplex>
        </serviceOptions>
      </serviceMainInfo>
      <serviceNetInfo>
        <inet>
          <ipv4Addrs> 169.254.10.10 (0xA9FE0A0A) </ipv4Addrs>
          <hostname> printer1.local. </hostname>
        </inet>
        <protocol>
          <location> acme:printdirect </location>
          <transPorts> tcp/9200 (0x000623F0) </transPorts>
        </protocol>
        <protocol>
          <location> lpr </location>
          <transPorts> tcp/515 (0x00060203) </transPorts>
        </protocol>
      </serviceNetInfo>
      <serviceAddInfo>
        <url> http://printer.local/index.html </url>
        <vendor> Acme Corporation </vendor>
        <vendorURL> http://www.acme.com/index.html </vendorURL>
        <model> Acme Matrix Printer 1000 </model>
        <modelURL> http://www.acme.com/printers/mp1000.html
        </modelURL>
        <version> 3.2 </version>
        <location> Basement </location>
      </serviceAddInfo>
    </service>
    <registerState>
      <selectState>
        <resources> 1 </resources>
        <workload> 0.0 </workload>
      </selectState>
    </registerState>
    <registerInfo>
      <lifetime> 2 </lifetime>
      <selectInfo>
        <policies> Max Priority (0x0003),
                   Least Used (0x0004)
        </policies>
        <priority> 1 </priority>
        <weight> 1.0 </weight>
      </selectInfo>
```

```
    </registerInfo>
    <updateInfo>
      <minLife> 3600 </minLife>
      <maxLife> 36000 </maxLife>
    </updateInfo>
  </registerService>
</xsrpv1>
```

- Mensaje de respuesta:

```
<xsrpv1>
 <header>
   <xid> 0x028B00E4 </xid>
   <realm>
     <domain> it.uc3m.es </domain>
     <scope> DEFAULT </scope>
   </realm>
   <source>
     <id> 3f44ce10-29fd-48dd-85aa-c7882f4e1c5e
          (0x3F44CE1029FD48DD85AAC7882F4E1C5E)
     </id>
   </source>
   <destination>
     <id> 77740ccb-6a9e-4572-95be-d701c9db104a
          (0x77740CCB6A9E457295BED701C9DB104A)
     </id>
   </destination>
 </header>
 <error>
   <errCode> SERVICE_COLLISION (0x000a0001) </errCode>
   <errName> SERVICE_COLLISION_ERROR </errName>
   <service>
     <id> aaaaaaaa-aaaa-4aaa-8aaa-aaaaaaaaaaaa
          (0xAAAAAAAAAAAA4AAA8AAAAAAAAAAAAAAA)
     </id>
   </service>
 </error>
</xsrpv1>
```

- Resultado: CORRECTO. Si realizamos ahora una consulta utilizando el protocolo XSLP, obtenemos como respuesta que no hay ningún servicio registrado.

Caso de Prueba 3

- Descripción: En este caso de prueba, se expone el funcionamiento del protocolo cuando se pretende registrar un servicio que posee una política de selección incompatible con la política de selección de otro servicio ya registrado del mismo tipo del que se pretende registrar.

  Para la realización de esta prueba se pasa como parámetro de entrada un documento .xml que contiene la información del servicio que se pretende registrar. Se debe recibir como respuesta al mensaje XSRP con la operación `registerService`, un mensaje XSRP con una operación de error, la cual indica que se ha producido un error por la utilización de políticas incompatibles.

- Mensaje de Petición:

```
<xsrpv1>
 <header>
   <xid> 0xF1CA2D7B </xid>
   <realm>
     <domain> it.uc3m.es </domain>
     <scope> DEFAULT </scope>
   </realm>
   <source>
     <id> 77740ccb-6a9e-4572-95be-d701c9db104a
          (0x77740CCB6A9E457295BED701C9DB104A)
     </id>
   </source>
   <destination>
     <id> e7b8f5bf-fa0a-4b01-b79c-06bcfea6e175
          (0xE7B8F5BFFA0A4B01B79C06BCFEA6E175)
     </id>
   </destination>
 </header>
 <registerService>
   <realmTarget>
     <scopesMap> true </scopesMap>
   </realmTarget>
   <service>
     <id> aaaaaaaa-aaaa-4aaa-8aaa-aaaaaaaaaaaa
          (0xAAAAAAAAAAAA4AAA8AAAAAAAAAAAAAAA)
     </id>
     <serviceState>
       <stateTimestamp> -982789859 </stateTimestamp>
```

```
  <selectState>
    <resources> 3 </resources>
    <workload> 0.0 </workload>
  </selectState>
</serviceState>
<serviceMainInfo>
  <serviceType>
    <type> printer </type>
  </serviceType>
  <alias> Old printer </alias>
  <selectInfo>
    <policies> Weighted Round Robin (0x0007),
               Least Used (0x0004)
    </policies>
    <priority> 5 </priority>
    <weight> 1.0 </weight>
  </selectInfo>
  <serviceOptions>
    <color ext="true" c="true"
           type="boolean"> false
    </color>
    <duplex ext="true" c="true"
            type="boolean"> false
    </duplex>
  </serviceOptions>
</serviceMainInfo>
<serviceNetInfo>
  <inet>
    <ipv4Addrs> 169.254.10.10 (0xA9FE0A0A) </ipv4Addrs>
    <hostname> printer1.local. </hostname>
  </inet>
  <protocol>
    <location> acme:printdirect </location>
    <transPorts> tcp/9200 (0x000623F0) </transPorts>
  </protocol>
  <protocol>
    <location> lpr </location>
    <transPorts> tcp/515 (0x00060203) </transPorts>
  </protocol>
</serviceNetInfo>
<serviceAddInfo>
  <url> http://printer.local/index.html </url>
  <vendor> Acme Corporation </vendor>
  <vendorURL> http://www.acme.com/index.html
```

```
        </vendorURL>
        <model> Acme Matrix Printer 1000 </model>
        <modelURL> http://www.acme.com/printers/mp1000.html
        </modelURL>
        <version> 3.2 </version>
        <location> Basement </location>
      </serviceAddInfo>
    </service>
    <registerState>
      <selectState>
        <resources> 1 </resources>
        <workload> 0.0 </workload>
      </selectState>
    </registerState>
    <registerInfo>
      <lifetime> 2 </lifetime>
      <selectInfo>
        <policies> Weighted Round Robin (0x0007),
                   Least Used (0x0004)
        </policies>
        <priority> 1 </priority>
        <weight> 1.0 </weight>
      </selectInfo>
    </registerInfo>
    <updateInfo>
      <minLife> 3600 </minLife>
      <maxLife> 36000 </maxLife>
    </updateInfo>
  </registerService>
</xsrpv1>
```

- Mensaje de Respuesta:

```
<xsrpv1>
 <header>
   <xid> 0xB20C89EF </xid>
   <realm>
     <domain> it.uc3m.es </domain>
     <scope> DEFAULT </scope>
   </realm>
   <source>
     <id> e7b8f5bf-fa0a-4b01-b79c-06bcfea6e175
          (0xE7B8F5BFFA0A4B01B79C06BCFEA6E175)
     </id>
```

```
    </source>
    <destination>
      <id> 77740ccb-6a9e-4572-95be-d701c9db104a
           (0x77740CCB6A9E457295BED701C9DB104A)
      </id>
    </destination>
  </header>
  <error>
    <errCode> INCOMPATIBLE_POLICY (0x000a0004) </errCode>
    <errName> INCOMPATIBLE_POLICY_ERROR </errName>
    <service>
      <id> aaaaaaaa-aaaa-4aaa-8aaa-aaaaaaaaaaaa
           (0xAAAAAAAAAAAA4AAA8AAAAAAAAAAAAAAA)
      </id>
    </service>
    <policies> Max Priority (0x0003),
               Least Used (0x0004)
    </policies>
  </error>
</xsrpv1>
```

- Resultado: CORRECTO. Al igual que en el caso anterior, tras la realización de la consulta utilizando XSLP, el resultado también es que no existe ningún servicio registrado en el Agente de Directorio (DA). En el mensaje de respuesta, además de mostrar el error que se ha producido, en este caso de política incompatible, se muestra el id del servicio que ha producido el error al intentar ser registrado, y la política que tienen los servicios registrados de ese tipo para facilitar un futuro registro evitando este error.

7.2. Pruebas del caso de uso Update Service

Caso de Prueba 4

- Descripción: En esta prueba se pretende actualizar un servicio registrado previamente. Cuando se realizó el registro, el alias de éste era "Old printer", pero ahora el alias ha cambiado y es "MODIFICADO". Debido a ésto se comprobará que el servicio ha cambiado y se procederá a la actualización del mismo.

  En este caso, su parámetro de entrada es un documento .xml con los datos del servicio, en este caso con el campo alias cambiado respecto al servicio cuando éste fue registrado.

Como resultado a esta petición, la cual es un mensaje XSRP con la operación `updateService`, es otro mensaje XSRP con la operación XSRP con la operación `updateServiceAck` que confirma la actualización del servicio.

- Parámetro de Entrada

```
<service>
  <id> aaaaaaaa-aaaa-4aaa-8aaa-aaaaaaaaaaaa
       (0xAAAAAAAAAAAA4AAA8AAAAAAAAAAAAAAA)
  </id>
  <serviceState>
    <stateTimestamp> 0 </stateTimestamp>
    <selectState>
      <workload> 0 </workload>
      <mainInfoSeqNum> 1 </mainInfoSeqNum>
    </selectState>
  </serviceState>
  <serviceMainInfo>
    <serviceType>
      <type> printer </type>
    </serviceType>
    <alias> MODIFICADO </alias>
    <selectInfo>
      <policies> Max Priority (0x0003),
                 Least Used (0x0004)
      </policies>
      <priority> 5 </priority>
    </selectInfo>
    <serviceOptions>
      <color  ext="true" c="true" e="false"
              type="boolean"> false
      </color>
      <duplex ext="true" c="true" e="false"
              type="boolean"> false
      </duplex>
    </serviceOptions>
  </serviceMainInfo>
  <serviceNetInfo>
    <inet>
      <ipv4Addrs> 169.254.10.10 (0xA9FE0A0A) </ipv4Addrs>
      <hostname> printer1.local. </hostname>
    </inet>
    <protocol>
```

```
        <protoName> acme:printdirect </protoName>
        <transPorts> tcp/9200 (0x000623F0) </transPorts>
      </protocol>
      <protocol>
        <protoName> lpr </protoName>
        <transPorts> tcp/515 (0x00060203) </transPorts>
      </protocol>
    </serviceNetInfo>
    <serviceAddInfo>
      <url> http://printer.local/index.html </url>
      <vendor> Acme Corporation </vendor>
      <vendorURL> http://www.acme.com/index.html </vendorURL>
      <model> Acme Matrix Printer 1000 </model>
      <modelURL> http://www.acme.com/printers/mp1000.html
      </modelURL>
      <version> 3.2 </version>
      <location> Basement </location>
    </serviceAddInfo>
  </service>
```

- Mensaje de Petición:

```
 <xsrpv1>
  <header>
    <xid> 0x74D93D79 </xid>
    <realm>
      <domain> it.uc3m.es </domain>
      <scope> DEFAULT </scope>
    </realm>
    <source>
      <id> 77740ccb-6a9e-4572-95be-d701c9db104a
           (0x77740CCB6A9E457295BED701C9DB104A)
      </id>
    </source>
    <destination>
      <id> 699330d1-29ac-48b1-a101-39a440c78069
           (0x699330D129AC48B1A10139A440C78069)
      </id>
    </destination>
  </header>
  <updateService>
    <realmTarget>
      <scopesMap> true </scopesMap>
```

```
    </realmTarget>
    <service>
      <id> aaaaaaaa-aaaa-4aaa-8aaa-aaaaaaaaaaaa
           (0xAAAAAAAAAAAA4AAA8AAAAAAAAAAAAAAA)
      </id>
      <serviceState>
        <stateTimestamp> 1258903469306 </stateTimestamp>
        <selectState>
          <workload> 0.0 </workload>
          <mainInfoSeqNum> 1 </mainInfoSeqNum>
        </selectState>
      </serviceState>
      <serviceMainInfo>
        <serviceType>
          <type> printer </type>
        </serviceType>
        <alias> MODIFICADO </alias>
        <selectInfo>
          <policies> Max Priority (0x0003),
                     Least Used (0x0004)
          </policies>
          <priority> 5 </priority>
          <weight> 1.0 </weight>
        </selectInfo>
        <serviceOptions>
          <color ext="true" c="true"
                 type="boolean"> false
          </color>
          <duplex ext="true" c="true"
                  type="boolean"> false
          </duplex>
        </serviceOptions>
      </serviceMainInfo>
    </service>
    <registerState>
    </registerState>
    <registerInfo>
      <lifetime> 3600000 </lifetime>
    </registerInfo>
    <updateInfo>
      <minLife> 3600 </minLife>
      <maxLife> 36000 </maxLife>
    </updateInfo>
  </updateService>
```

```
</xsrpv1>
```

- Mensaje de Respuesta:

```
<xsrpv1>
 <header>
   <xid> 0x74D93D79 </xid>
   <realm>
     <domain> it.uc3m.es </domain>
     <scope> DEFAULT </scope>
   </realm>
   <source>
     <id> 699330d1-29ac-48b1-a101-39a440c78069
          (0x699330D129AC48B1A10139A440C78069)
     </id>
   </source>
   <destination>
     <id> 77740ccb-6a9e-4572-95be-d701c9db104a
          (0x77740CCB6A9E457295BED701C9DB104A)
     </id>
   </destination>
 </header>
 <updateServiceAck>
   <service>
     <id> aaaaaaaa-aaaa-4aaa-8aaa-aaaaaaaaaaaa
          (0xAAAAAAAAAAAA4AAA8AAAAAAAAAAAAAAA)
     </id>
   </service>
   <updateInfo>
     <minLife> 335 </minLife>
     <maxLife> 3350 </maxLife>
   </updateInfo>
 </updateServiceAck>
</xsrpv1>
```

No se obtiene un parámetro de salida propiamente dicho. En esta prueba, la operación concluye de manera satisfactoria por lo que el servicio correspondiente, cuyos datos se aportan en el parámetro de entrada en formato .xml, es actualizado.

- Resultado: CORRECTO

Caso de Prueba 5

- Descripción: En este caso de prueba, se intenta actualizar un servicio que no ha sido registrado con anterioridad. Como parámetro de entrada tenemos un documento .xml con la información del servicio que se quiere actualizar, pero que no había sido registrado previamente.

  Como resultado al envío de un mensaje XSRP con la operación `updateService`, donde el servicio no había sido registrado, es otro mensaje XSRP con una operación de error que contenga el error SERVICE_NOT_FOUND.

- Mensaje de Petición:

```
<xsrpv1>
 <header>
   <xid> 0xB5BD2346 </xid>
   <realm>
     <domain> it.uc3m.es </domain>
     <scope> DEFAULT </scope>
   </realm>
   <source>
     <id> 77740ccb-6a9e-4572-95be-d701c9db104a
          (0x77740CCB6A9E457295BED701C9DB104A)
     </id>
   </source>
   <destination>
     <id> 399d8625-fe89-4d81-818b-3e89e9d3e99b
          (0x399D8625FE894D81818B3E89E9D3E99B)
     </id>
   </destination>
 </header>
 <updateService>
   <realmTarget>
     <scopesMap> true </scopesMap>
   </realmTarget>
   <service>
     <id> aaaaaaaa-aaaa-4aaa-8aaa-aaaaaaaaaaaa
          (0xAAAAAAAAAAAA4AAA8AAAAAAAAAAAAAAA)
     </id>
     <serviceState>
       <stateTimestamp> 1258907838869 </stateTimestamp>
       <selectState>
         <workload> 0.0 </workload>
```

```
            <mainInfoSeqNum> 1 </mainInfoSeqNum>
          </selectState>
        </serviceState>
        <serviceMainInfo>
          <serviceType>
            <type> printer </type>
          </serviceType>
          <alias> MODIFICADO </alias>
          <selectInfo>
            <policies> Max Priority (0x0003),
                       Least Used (0x0004)
            </policies>
            <priority> 5 </priority>
            <weight> 1.0 </weight>
          </selectInfo>
          <serviceOptions>
            <color ext="true" c="true"
                   type="boolean"> false
            </color>
            <duplex ext="true" c="true"
                    type="boolean"> false
            </duplex>
          </serviceOptions>
        </serviceMainInfo>
      </service>
      <registerState>
      </registerState>
      <registerInfo>
        <lifetime> 3600000 </lifetime>
      </registerInfo>
      <updateInfo>
        <minLife> 3600 </minLife>
        <maxLife> 36000 </maxLife>
      </updateInfo>
    </updateService>
  </xsrpv1>
```

- Mensaje de Respuesta:

```
<xsrpv1>
  <header>
    <xid> 0xB5BD2346 </xid>
```

```
      <realm>
        <domain> it.uc3m.es </domain>
        <scope> DEFAULT </scope>
      </realm>
      <source>
        <id> 399d8625-fe89-4d81-818b-3e89e9d3e99b
             (0x399D8625FE894D81818B3E89E9D3E99B)
        </id>
      </source>
      <destination>
        <id> 77740ccb-6a9e-4572-95be-d701c9db104a
             (0x77740CCB6A9E457295BED701C9DB104A)
        </id>
      </destination>
    </header>
    <error>
      <errCode> SERVICE_NOT_FOUND (0x000A0002) </errCode>
      <errName> SERVICE_NOT_FOUND_ERROR </errName>
      <service>
        <id> aaaaaaaa-aaaa-4aaa-8aaa-aaaaaaaaaaaa
             (0xAAAAAAAAAAAA4AAA8AAAAAAAAAAAAAAA)
        </id>
      </service>
    </error>
  </xsrpv1>
```

- Resultado: CORRECTO.

Caso de Prueba 6

- Descripción: En esta prueba se pretende actualizar un servicio, en cuya petición está incluida un elemento *Selection Information* que contiene una política de selección incompatible con la que presenta el servicio que ha sido registrado anteriormente.

  Como parámetro de entrada se presenta un documento .xml con la información del servicio donde podemos ver que el campo *policies* tiene una política que, en este caso, es incompatible con la política que presentaba el servicio registrado.

  Como respueta al mensaje XSRP con la operación `updateService` en esta circunstancia descrita anteriormente se recibe un mensaje XSRP

con una operación de error donde se indica un problema de incompatibilidad con la política de selección del servicio que se pretende actualizar.

- Parámetro de Entrada:

```
<service>
 <id> aaaaaaaa-aaaa-4aaa-8aaa-aaaaaaaaaaaa
      (0xAAAAAAAAAAAA4AAA8AAAAAAAAAAAAAA)
 </id>
 <serviceState>
   <stateTimestamp> 0 </stateTimestamp>
   <selectState>
     <workload> 0 </workload>
     <mainInfoSeqNum> 1 </mainInfoSeqNum>
   </selectState>
 </serviceState>
 <serviceMainInfo>
   <serviceType>
     <type> printer </type>
   </serviceType>
   <alias> MODIFICADO </alias>
   <selectInfo>
     <policies> Weighted Round Robin (0x0007),
                Least Used (0x0004)
     </policies>
     <priority> 5 </priority>
   </selectInfo>
   <serviceOptions>
     <color  ext="true" c="true" e="false"
             type="boolean"> false
     </color>
     <duplex ext="true" c="true" e="false"
             type="boolean"> false
     </duplex>
   </serviceOptions>
 </serviceMainInfo>
 <serviceNetInfo>
   <inet>
     <ipv4Addrs> 169.254.10.10 (0xA9FE0A0A) </ipv4Addrs>
     <hostname> printer1.local. </hostname>
   </inet>
   <protocol>
     <protoName> acme:printdirect </protoName>
```

```
        <transPorts> tcp/9200 (0x000623F0) </transPorts>
      </protocol>
      <protocol>
        <protoName> lpr </protoName>
        <transPorts> tcp/515 (0x00060203) </transPorts>
      </protocol>
    </serviceNetInfo>
    <serviceAddInfo>
      <url> http://printer.local/index.html </url>
      <vendor> Acme Corporation </vendor>
      <vendorURL> http://www.acme.com/index.html </vendorURL>
      <model> Acme Matrix Printer 1000 </model>
      <modelURL> http://www.acme.com/printers/mp1000.html
      </modelURL>
      <version> 3.2 </version>
      <location> Basement </location>
    </serviceAddInfo>
  </service>
```

- Mensaje de Petición:

```
<xsrpv1>
  <header>
    <xid> 0x97DB7511 </xid>
    <realm>
      <domain> it.uc3m.es </domain>
      <scope> DEFAULT </scope>
    </realm>
    <source>
      <id> 77740ccb-6a9e-4572-95be-d701c9db104a
           (0x77740CCB6A9E457295BED701C9DB104A)
      </id>
    </source>
    <destination>
      <id> 399d8625-fe89-4d81-818b-3e89e9d3e99b
           (0x399D8625FE894D81818B3E89E9D3E99B)
      </id>
    </destination>
  </header>
  <updateService>
    <realmTarget>
      <scopesMap> true </scopesMap>
    </realmTarget>
```

```
    <service>
      <id> aaaaaaaa-aaaa-4aaa-8aaa-aaaaaaaaaaaa
           (0xAAAAAAAAAAAA4AAA8AAAAAAAAAAAAAAA)
      </id>
      <serviceState>
        <stateTimestamp> 1258908118737 </stateTimestamp>
        <selectState>
          <workload> 0.0 </workload>
          <mainInfoSeqNum> 1 </mainInfoSeqNum>
        </selectState>
      </serviceState>
      <serviceMainInfo>
        <serviceType>
          <type> printer </type>
        </serviceType>
        <alias> MODIFICADO </alias>
        <selectInfo>
          <policies> Weighted Round Robin (0x0007),
                     Least Used (0x0004)
          </policies>
          <priority> 5 </priority>
          <weight> 1.0 </weight>
        </selectInfo>
        <serviceOptions>
          <color ext="true" c="true"
                 type="boolean"> false
          </color>
          <duplex ext="true" c="true"
                  type="boolean"> false
          </duplex>
        </serviceOptions>
      </serviceMainInfo>
    </service>
    <registerState>
    </registerState>
    <registerInfo>
      <lifetime> 3600000 </lifetime>
    </registerInfo>
    <updateInfo>
      <minLife> 3600 </minLife>
      <maxLife> 36000 </maxLife>
    </updateInfo>
  </updateService>
</xsrpv1>
```

- Mensaje de Respuesta:

```
<xsrpv1>
  <header>
    <xid> 0x97DB7511 </xid>
    <realm>
      <domain> it.uc3m.es </domain>
      <scope> DEFAULT </scope>
    </realm>
    <source>
      <id> 399d8625-fe89-4d81-818b-3e89e9d3e99b
           (0x399D8625FE894D81818B3E89E9D3E99B)
      </id>
    </source>
    <destination>
      <id> 77740ccb-6a9e-4572-95be-d701c9db104a
           (0x77740CCB6A9E457295BED701C9DB104A)
      </id>
    </destination>
  </header>
  <error>
    <errCode> INCOMPATIBLE_POLICY (0x000A0004) </errCode>
    <errName> INCOMPATIBLE_POLICY_ERROR </errName>
    <service>
      <id> aaaaaaaa-aaaa-4aaa-8aaa-aaaaaaaaaaaa
           (0xAAAAAAAAAAAA4AAA8AAAAAAAAAAAAAAA)
      </id>
    </service>
    <policies> Max Priority (0x0003),
               Least Used (0x0004)
    </policies>
  </error>
</xsrpv1>
```

- Resultado: CORRECTO

7.3. Pruebas del caso de uso Deregister Service

Caso de Prueba 7

- Descripción: En este caso se pretende borrar un servicio que ha sido previamente registrado. En este caso, el parámetro de entrada es un documento .xml donde se encuentra la información del servicio que se pretende borrar.

Para la realización de esta prueba, previamente se ha registrado este servicio que se pretende borrar. La respuesta a la peticion XSRP con la operación `deregisterService`, es un mensaje XSRP con la operación `deregisterServiceAck` que confirma que se ha borrado correctamente el servicio.

- Mensaje de Petición:

```
<xsrpv1>
  <header>
    <xid> 0x35A31853 </xid>
    <realm>
      <domain> it.uc3m.es </domain>
      <scope> DEFAULT </scope>
    </realm>
    <source>
      <id> 77740ccb-6a9e-4572-95be-d701c9db104a
           (0x77740CCB6A9E457295BED701C9DB104A)
      </id>
    </source>
    <destination>
      <id> 3f44ce10-29fd-48dd-85aa-c7882f4e1c5e
           (0x3F44CE1029FD48DD85AAC7882F4E1C5E)
      </id>
    </destination>
  </header>
  <deregisterService>
    <service>
      <id> aaaaaaaa-aaaa-4aaa-8aaa-aaaaaaaaaaaa
           (0xAAAAAAAAAAAA4AAA8AAAAAAAAAAAAAAA)
      </id>
      <serviceMainInfo>
        <serviceType>
          <type> printer </type>
        </serviceType>
      </serviceMainInfo>
    </service>
  </deregisterService>
</xsrpv1>
```

- Mensaje de Respuesta:

```
 <xsrpv1>
  <header>
```

```
    <xid> 0x302BAB8F </xid>
    <realm>
      <domain> it.uc3m.es </domain>
      <scope> DEFAULT </scope>
    </realm>
    <source>
      <id> 3f44ce10-29fd-48dd-85aa-c7882f4e1c5e
           (0x3F44CE1029FD48DD85AAC7882F4E1C5E)
      </id>
    </source>
    <destination>
      <id> 77740ccb-6a9e-4572-95be-d701c9db104a
           (0x77740CCB6A9E457295BED701C9DB104A)
      </id>
    </destination>
  </header>
  <deregisterServiceAck>
    <service>
      <id> aaaaaaaa-aaaa-4aaa-8aaa-aaaaaaaaaaaa
           (0xAAAAAAAAAAAA4AAA8AAAAAAAAAAAAAAA)
      </id>
    </service>
  </deregisterServiceAck>
</xsrpv1>
```

- Resultado: CORRECTO. Para comprobar que se ha realizado correctamente el borrado en el Agente de Directorio (DA) de un servicio previamente registrado, se ha realizado una consulta utilizando el protocolo XSLP donde podemos ver que no hay ningún servicio registrado en el DA.

Caso de Prueba 8

- Descripción: En esta prueba, se pretende borrar un servicio, pero el id de éste no coincide con ninguno de los servicios previamente registrados. En este caso, como respuesta a la petición XSRP con la operación `deregisterService` se envía un mensaje XSRP con la operación de error, el cual se produce al no encontrar el servicio que se pretende borrar, al no haber sido registrado previamente.

- Parámetros de Entrada:

```
<xsrpv1>
  <header>
    <xid> 0x302BAB8F </xid>
```

```
    <realm>
      <domain> it.uc3m.es </domain>
      <scope> DEFAULT </scope>
    </realm>
    <source>
      <id> 77740ccb-6a9e-4572-95be-d701c9db104a
           (0x77740CCB6A9E457295BED701C9DB104A)
      </id>
    </source>
    <destination>
      <id> 64600659-b314-47f1-9632-a210af5a52ed
           (0x64600659B31447F19632A210AF5A52ED)
      </id>
    </destination>
  </header>
  <deregisterService>
    <service>
      <id> aaaaaaaa-aaaa-4aaa-8aaa-aaaaaaaaaaaa
          (0xAAAAAAAAAAAA4AAA8AAAAAAAAAAAAAAA)
      </id>
      </service>
  </deregisterService>
</xsrpv1>
```

- Mensaje de Respuesta:

```
<xsrpv1>
  <header>
    <xid> 0x35A31853 </xid>
    <realm>
      <domain> it.uc3m.es </domain>
      <scope> DEFAULT </scope>
    </realm>
    <source>
      <id> 64600659-b314-47f1-9632-a210af5a52ed
           (0x64600659B31447F19632A210AF5A52ED)
      </id>
    </source>
    <destination>
      <id> 77740ccb-6a9e-4572-95be-d701c9db104a
           (0x77740CCB6A9E457295BED701C9DB104A)
      </id>
```

```
    </destination>
  </header>
  <error>
    <errCode> UNKNOWN_SERVICE_ID (0x00000004) </errCode>
    <errName> UNKNOWN_SERVICE_ID_ERROR </errName>
    <service>
      <id> aaaaaaaa-aaaa-4aaa-8aaa-aaaaaaaaaaaa
           (0xAAAAAAAAAAAA4AAA8AAAAAAAAAAAAAAA)
      </id>
    </service>
  </error>
</xsrpv1>
```

- Resultado: CORRECTO. En este caso, tras la realización de la consulta utilizando el protocolo XSLP podemos ver que no se ha borrado ningún servicio y se encuentran registrados los mismos que lo estaban hasta ese momento.

A continuación se muestra una tabla donde se resumen los distintos casos de prueba anteriormente descritos y el resultado obtenido. En la tabla se muestra en primer lugar el identificador del caso de prueba, que coincidirá con lo anteriormente descrito. A continuación una breve descripción del caso de prueba y por último el resultado de la misma. Gracias a este conjunto de pruebas se demuestra que hemos conseguido la funcionalidad que se pretendía en un primer momento.

Id	Caso de Prueba	Resultado
1	Registrar un Servicio	CORRECTO
2	Registrar un Servicio que había sido registrado previamente	CORRECTO
3	Registrar un Servicio con una política incompatible	CORRECTO
4	Actualizar un Servicio	CORRECTO
5	Actualizar un Servicio que no había sido previamente registrado	CORRECTO
6	Actualizar un Servicio con una política incompatible	CORRECTO
7	Borrar un Servicio	CORRECTO
8	Borrar un Servicio que no había sido previamente registrado	CORRECTO

Además de estas pruebas, también se ha probado que todas estas operaciones se pueden realizar utilizando tanto el protocolo de transporte UDP como TCP. Para la realización de esta prueba se ha cambiado el orden de las preferencias de los protocolos de trasporte comentado en el capítulo anterior. Gracias a esta prueba se ha comprobado el correcto funcionamiento del protocolo utilizando ambos protocolos de trasporte.

Otra prueba realizada ha sido la retransmisión de mensajes cuando se produce un error tal como la ausencia de un DA al que dirigir las peticiones. En este caso se realizan hasta tres intentos y se lanza una excepción que informa de lo sucedido.

Además de esto, se ha procedido a realizar registros tanto con el parámetro opcional "lifetime" como sin él. En caso de que se proporcione este parámetro, ese valor se utilizará para determinar el tiempo de vida del servicio, si antes de ese tiempo no ha sido actualizado se procederá a su borrado. En caso de no proporcionarse dicho parámetro se asignará un valor por defecto. También se han realizado pruebas introduciendo como valor de dicho parámetro un número menor que cero y comprobando que la funcionalidad es la deseada, la cual ha sido descrita en el API de Service Agent.

También se ha probado que pasado el tiempo de vida del servicio, éste es borrado. Si ha sido actualizado, también lo habrá sido su tiempo de vida, pero una éste ha vencido se borrará.

Todas estas pruebas han producido un resultado correcto, lo que nos hace pensar que se controlan los principales errores que podían producirse y se ha alcanzado la funcionalidad que se pretendía en un primer momento.

Capítulo 8

Gestión del Proyecto

El objetivo de este capítulo es detallar la planificación temporal y los costes del proyecto. De este modo, se puede estimar el tiempo y coste de cada una de las etapas que lo conforman.

8.1. Planificación Temporal

En esta sección se pretende mostrar las distintas etapas que han tenido lugar para el desarrollo de este proyecto fin de carrera, así como el tiempo que se ha tardado en la realización de cada una de ellas.

A continuación se describirá en detalle cada una de las tareas que se han desarrollado, el tiempo que ha llevado su realización y una descripción lo más detallada posible de la tarea en sí.

8.1.1. Estudio de los Protocolos de Descubrimiento de Servicio

Antes de comenzar con el desarrollo del proyecto ha sido necesario familiarizarse con los distintos protocolos ya existentes que cubren las necesidades actuales de descubrimiento de servicio.

De este modo, analizando su metodología, las distintas ventajas e inconvenientes que aportan podemos comentar de un modo mas exhaustivo qué proporciona XSDF que no aportan los protocolos ya existentes.

Cabe destacar que como primera toma de contacto con la temática del proyecto, se realizó un trabajo dirigido consintente en el estudio de uno de los protocolos de descubrimiento de servicio con mayor presencia en el mercado actualmente como es *Universal Plug and Play* (UPnP).

Una vez se ha empezado con el estudio del resto de protocolos de descubrimiento de servicio, se inició la elaboración de la memoria. En particular se empieza a elaborar el estado del arte del proyecto. A partir de ahí, la elaboración de la memoria sigue de manera constante mientras se realiza el proyecto, para realizar la documentación al mismo tiempo que el desarrollo.

8.1.2. Análisis

En esta etapa el objetivo es estudiar la documentación proporcionada, como son los *draft* de los distintos protocolos que conforman XSDF, para su estudio detallado antes empezar el diseño e implementación. También se analiza en este momento el código proporcionado donde están implementados XBE32, el protocolo XSLP y la arquitectura general de XSDF.

Estudio de los protocolos XSDF

Esta parte concierne al estudio de los *draft* donde podemos conocer las distintas operaciones que llevan a cabo cada uno de ellos, el intercambio de mensajes necesario para llevarlas a cabo, los diferentes elementos que componen los mensajes, su codificación XBE32, etc.

Esto nos proporciona una visión general de todos los protocolos que conforman XSDF y cómo funcionan en conjunto, para luego poder centrarnos en XSRP.

Estudio del código proporcionado

En este útimo paso previo al comienzo del diseño y la implementación, se prentende familiarizarse con la arquitectura de la aplicación y la implementación del protocolo XSLP, que es necesaria para el correcto funcionamiento de XSRP.

Este apartado nos ha ayudado al diseño e implementación del protocolo, y su integración con el código existente, así como la asimilación de la arquitectura de XSDF que tiene bastante complejidad.

8.1.3. Diseño e Implementación

En esta etapa pasamos a diseñar e implementar el protocolo XSRP de modo que se integre con la arquitectura existente. El objetivo es implementar las tres operaciones que realiza el protocolo (Registrar un servicio, Actualizar un servicio y Borrar un servicio) integrándolo con el protocolo ya implementado y realizando la gestión de errores adecuada. De este modo se intentan

controlar los errores que se pueden producir, gestionarlos de manera adecuada e informar de forma precisa de lo acontecido en la realización de la operación.

8.1.4. Diseño e Implementación de Pruebas

Una vez realizada la implementación de cada una de las operaciones, se han realizado las pruebas que demuestren el buen funcionamiento de las mimas. Estas pruebas han consistido en la ejecución de un cliente en circunstancias previamente especificadas con el objetivo de encontrar errores que pudiesen darse cuando se utilice el protocolo.

Como se ha mostrado en el capítulo anterior dedicado a los casos de prueba que se han realizado para comprobar el buen funcionamiento del protocolo, se pretende englobar en estos casos de prueba los distintos casos de uso de la aplicación, de modo que queden cubiertos todas las posibles situaciones que pueden tener lugar cuando se utilice este protocolo.

8.1.5. Elaboración de la memoria

Esta tarea se ha comenzado a realizar cuando se empiezan a estudiar los distintos protocolos de descubrimiento de servicio utilizados en la actualidad. En ese momento se empieza a elaborar el estado del arte del proyecto y durante las distintas etapas se intenta documentar de manera paralela al desarrollo.

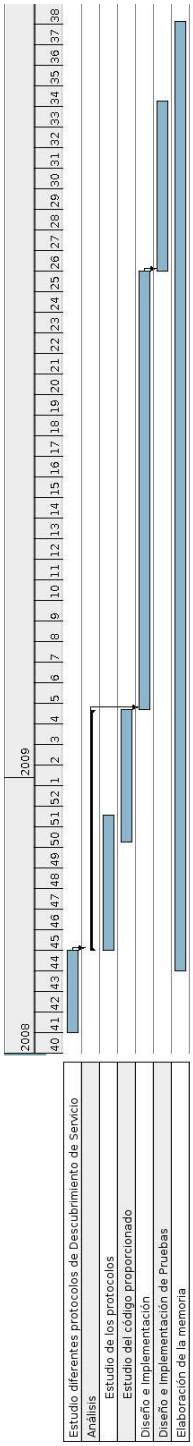


Figura 8.1: Diagrama de Gantt

8.2. Análisis de Costes

En este apartado se comentarán los costes de la realización del proyecto. Dichos costes se dividirán en tres grupo según la naturaleza de los mismos: Costes hardware, costes software y costes de personal, los cuales se mostrarán cada uno en su tabla correspondiente y, por último una tabla que nos indica el coste total, incluyendo todos los costes implicados.

Costes Hardware

Estos costes se refieren a la compra del material utilizado para el desarrollo del proyecto en todas sus etapas. El coste del ordenador asciende a 1.128'45 €. Aplicándole el 16 % de IVA correspondiente que una cantidad de 1309 €. Suponiendo un *tiempo de uso* de 4 años, el coste de amortización de un año sería de 327'25 €.

Componente Hardware	Coste
MacBook 4.1	327'25 €

Costes Software

En este caso, se comentan los costes derivados del software utilizado. Debido a la utilización de herramientas gratuitas se puede comprobar rápidamente observando la tabla a continuación que este coste será nulo.

Software	Coste
Java SDK 1.6	0 €
Netbeans IDE	0 €
Umbrello UML Modeller	0 €
DIA Diagrammer	0 €
Kile - Integrated LaTeX Environment	0 €

Costes de Personal

Para terminar, se expondrá el coste generado por el personal encargado del desarrollo, que en este caso se trata de dos personas (alumna y tutor). Se considerará un coste estimado de 65 euros/hora como salario de un Ingeniero Técnico de Telecomunicaciones, un coste de 95 euros/hora como salario de un Ingeniero de Telecomunicaciones (datos anteriormente publicados en el Colegio Oficial de Igenieros de Telecomunicación) y el número total de horas dedicadas de 500 y 20 respectivamente. Como resultado se muestra una tabla con estos datos.

Número de horas	Coste/hora	Coste total
500	65€/hora	32.500 €
20	95€/hora	1.900 €

Coste Total

Por último, se muestra una tabla con todos los costes anteriormente desglosados para poder mostrar el resultado final.

Tipo de Coste	Coste
Coste Hardware	327'25 €
Coste Software	0 €
Coste de Personal	34.400 €
Coste Total	**34.727'25 €**

Capítulo 9

Conclusiones y Trabajos Futuros

En este proyecto se buscaba desarrollar un protocolo ligero y extensible que formará parte de un conjunto de cuatro protocolos que conforma el eXtensible Service Discovery Framework (XSDF) para realizar las funciones de descubrimiento de servicio y balanceo de carga. Cada uno de estos cuatro protocolos realiza una función necesaria para llevar a cabo este cometido. Concretamente el protocolo desarrollado, el eXtensible Service Registration Protocol (XSRP), es el encargado del registro, borrado y actualización de los servicio de un Agente de Servicio en un Agente de Directorio.

Consideramos que este objetivo se ha cumplido, puesto que en este proyecto se ha realizado, después de un análisis riguroso, el diseño, la implementación y la validación tanto de un cliente como de un servidor XSRP.

La finalidad de XSDF es solventar los problemas tanto de descubrimiento de servicios como de reparto de carga, integrando en una arquitectura común ambas soluciones. XSDF hereda la arquitectura y nomenclatura de SLP, pero con la diferencia significativa de que XSDF está formado por un conjunto de cuatro protocolos simples en lugar de de un sólo protocolo más complejo.

Aunque esto pueda dar una imagen de mayor complejidad, esta separación propicia una arquitectua más sencilla. Esto también simplifica los agentes que forman parte de la arquitectura que sólo tendrán que implementar la parte cliente o servidora necesaria para la realización de su función.

Dada la magnitud y complejidad del proyecto han tenido lugar un conjunto de circunstancias adversas o problemáticas que ha sido necesario solventar para la realización del proyecto.

Una de las primeras dificultades a las que nos hemos visto expuestos es la necesidad de integrar nuestra implementación a un código ya existente de gran envergadura y con una arquitectura compleja. Esto ha propiciado una fase de análisis exhaustiva para familiarizarse con la arquitectura y posibilitar la realización de un diseño acertado fácilmente integrable en el código proporcionado.

Además de familiarizarse con la arquitectura, en la fase de análisis ha sido necesario un estudio riguroso del código proporcionado debido a las dependencias existentes con la implementación realizada.

El material suministrado para una primera toma de contacto con el proyecto han sido, además del mencionado código, los *draft* de los diferentes protocolos que conforman XSDF. Éstos dan información acerca del formato de los mensajes y los procedimientos a llevar a cabo para la realización de las distintas operaciones del protocolo.

Para la realización del diseño y la implementación del protocolo XSRP, una vez se ha realizado el estudio de los distintos *draft* y del código proporcionado, se ha respetado dicha arquitectura y se han utizado los métodos y clases genéricas para una mayor sencillez y transparencia.

Como se ha comentado en el capítulo 5, encargado de explicar el diseño del cliente y servidor XSRP, esta arquitectura está compuesta por diferentes capas, cada una de ellas encargada de una función concreta que repercute en el correcto funcionamiento del protocolo.

El agente que se encarga de realizar la función de servidor es el Agente de Directorio, mientras que el Agente de Servicio realiza la función de cliente. Es este último el encargado de enviar las distintas peticiones para la realización de las operaciones del protocolo mientras que el Agente de Directorio recibe esas operaciones, las gestiona realizando la operación pertinente y envía un mensaje de respuesta adecuada al cliente, dependiendo de si se ha realizado la operación correctamente o si ha ocurrido un error.

Debido a lo anteriormente comentado, este proyecto fin de carrera me ha permitido trabajar en un proyecto grande, partiendo de un código fruto de dos proyectos fin de carrera anteriores, con la dificultad que ello conlleva, y teniendo que integrar la implementación en una arquitectura compleja ya establecida.

Figura 9.1: Apple Bonjour

Además de esto, se ha procurado realizar un trabajo de documentación adecuado para facilitar futuros trabajos derivados de este proyecto. Para ello se ha aportado una gran cantidad de información tanto textual como gráfica haciendo especial hincapié en que se han utilizado diagramas tanto de carácter estático como de carácter dinámico con el fin de presentar de una manera más completa el trabajo realizado. Para esto se ha utilizado el *Unified Modeling Language (UML)* el cual apenas se había utilizado durante la carrera, por lo que fue necesario recabar información, en especial para la realización de los diagramas de secuencia.

A lo largo de este documento se ha comentado la funcionalidad, arquitectura, presencia en el mercado, implementaciones, etc. de diversos protocolos encargados del descubrimiento de servicios. Se ha podido comprobar la importancia de tener detrás una empresa con una presencia destacada para que la vida de dichos protocolos sea larga y próspera.

Esta reflexión está claramente referida a la situación que podemos apreciar hoy en día en el mercado, donde vemos que los protocolos de los que hemos hablado que están actualmente operativos son UPnP, respaldado por *Microsoft*, y la implementación de Zeroconf de Apple, *Bonjour*, que estos utilizan en aplicaciones tan populares actualmente como iTunes en el caso de Apple o en productos orientados al consumo como la XBOX 360 en el caso de *Microsoft*. Otros protocolos no han tenido esa suerte y se han quedado por el camino, como podría ser el caso de SALUDATION, JINI,... por citar

Figura 9.2: UPnP utilizado por la XBOX 360

alguno.

En capítulos anteriores, se ha pretendido describir la situación actual en lo que concierne a este campo y las aportaciones que hace XSDF al mismo.

El conjunto de protocolos que conforma XSDF pretende aprovechar la arquitectura que utiliza SLP para la realización del descubrimiento y de este modo también conseguir esa escalabilidad que permite al protocolo ser útil no sólo en redes pequeñas tipo LAN, sino también en grandes redes corporativas.

A pesar de heredar de SLP la arquitectura y nomenclatura, XSDF se componen de cuatro protocolos mas sencillos que realizan distintas comunicaciones, a diferencia de SLP, donde el mismo protocolo realiza todas estas comunicaciones. Además de realizar esta funcionalidad XSDF combina la capacidad de resolver problemas de descubrimiento y de reparto de carga.

Centrándonos en XSRP, cabe destacar que es un protocolo ligero, encargado del registro de servicios en los DA por parte de los SA. En todo momento se intenta no sobrecargar la red con mensajes innecesarios o información no relevante. Su función es necesaria para el buen funcionamiento del conjunto de los protocolos, la presencia de un repositorio central debidamente actualizado es fundamental, y esto es posible gracias a XSRP.

9.1. Trabajos Futuros

Cabe destacar que aunque este proyecto fin de carrera esté delimitado, existe un conjunto de actividades que sería deseable realizar en un futuro

cercano con el fin de completar lo aquí expuesto.

Parece bastante sencillo pensar que los trabajos futuros más inmediatos después del diseño e implementación de XSRP y pudiendo disponer de la implementación de XSLP, sería el diseño e implementación de los otros dos procolos que conforman XSDF como son XSSP y XSTP. Obviamente, el diseño de éstos debe hacerse basándose en la arquitectura ya existente para una fácil integración con lo ya implementado.

Una vez desarrollados, integrar los cuatro protocolos y realizar una fase de pruebas rigurosa para comprobar que realizan sus funciones correctamente y que todas las posibles situaciones han sido tenidas en cuenta y no afectarán al correcto funcionamiento del conjunto de protocolos. Para la realización de estas pruebas sería deseable montar un escenario completo donde se pueda confirmar el correcto funcionamiento de todos los protocolos y la realización de todas las pruebas necesarias para garantizar su correcta implementación e integración.

Otro de los aspectos que sería aconsejable abordar en un futuro sería proporcionar independencia de protocolo a las distintos agentes que conforman la arquitectura con el fin de que puedan interactuar no sólo con los protocolos de descubrimiento de servicio aquí descritos, sino que puedan ser utilizados cualquiera de los disponibles en el momento actual o en el futuro.

Además de esto, sería posible pensar en la implementación en C de los cuatro protocolos que componen XSDF. Teniendo en cuenta que en otro proyecto fin de carrera se ha implementando en C la librería XBE32, el siguiente paso sería empezar a implementar la arquitectura de XSDF y los distintos protocolos que lo conforman.

La implementación en C de los distintos protocolos que conforman XSDF permitirá que la adopción de estos protocolos por dispositivos limitados sea factible debido a que se necesitarían menos recursos dada la eficiencia del lenguaje.

Bibliografía

[1] Sun Microsystems. *Technical White Paper. Jini Architectual Overview.* Disponible en http://sun.com/jini [consulta: 30/03/09]

[2] UPnP Forum. http://upnp.org [consulta 01/04/09]

[3] HELAL, Sumi. *Standards for Service Discovery and Delivery.*en Pervasive Computing, IEEE. 2002. ISSN: 1536-1268

[4] Zero Configuration Networking. [online] http://www.zeroconf.org [consulta: 25/03/09]

[5] Apple Inc. [online] http://www.apple.com [consulta: 20/03/09]

[6] E. Guttman, C. Perkins. *"Service Location Protocol, Version 2"* IETF. RFC 2608. Junio 1999; Disponible en http://rfc-editor.org/rfc/rfc2608.txt [consulta 10/04/09]

[7] Stevens, W. Richard. *TCP/IP Illustrated.* Addison-Wesley. Reading (Massachusetts).

[8] *Extensible Markup Language (XML) 1.0 (Fifth Edition)* Disponible en http://www.w3.org/TR/REC-xml/ [consulta 10/04/09]

[9] *HTML 4.01 Specification.* Disponible en http://www.w3.org/TR/REC-html40/ [consulta 10/04/09]

[10] R. Droms. *Dynamic Host Configuration Protocol.* IETF. RFC 2131. Marzo 1997. Disponible en http://www.rfc-editor.org/rfc/rfc2131.txt

[11] *DNS Related RFCs.* [online] http://www.dns.net/dnsrd/rfc/. [consulta 12/04/09]

[12] *HTTP - Hypertext Transfer Protocol.* [online] http://www.w3.org/protocols [consulta 10/03/09]

[13] *Simple Service Discovery Protocol/1.0*[online] http://tools.ietf.org/html/draft-cai-ssdp-v1-00

[14] *General Event Notification Architecture Base.* [online] http://tools.ietf.org/html/draft-cohen-gena-p-base-01. [consulta 10/03/09]

[15] *SOAP Specifications.* [online] http://www.w3.org/TR/soap/ [consulta 11/03/09]

[16] *Wi-Fi Alliance.* [online] http://wi-fi.org [consulta: 30/06/09]

[17] Avahi. [online] http://avahi.org [consulta: 20/03/09]

[18] GNU/LINUX. [online] http://www.debian.org [consulta:20/03/09]

[19] Ubuntu. [online] http://www.ubuntu.com [consulta:20/03/09]

[20] Jae Woo Lee; Schulzrinne, H.; Kellerer, W.; Despotovic, Z.; *z2z: Discovering Zeroconf Services Beyond Local Link* en Globecom Workshops, 2007 IEEE. Disponible en http://www1.cs.columbia.edu/ hgs/papers/z2z-paper-1.1.pdf [consulta: 30/03/09]

[21] Bonjour for Windows.
[online] http://www.apple.com/support/downloads/bonjourforwindows [consulta 30/03/09]

[22] DNS-SD service types. [online] http://www.dns-sd.org/ServiceType.html

[23] Zero Configuration Networking (zeroconf) Working Group charter. [online] http://ietf.org/html.charters/OLD/zeroconf-charter.html [consulta 30/03/09]

[24] Cheshire S; Aboba B; Guttman B; *Dynamic configuration of IPv4 link-local address*, RFC 3927, May 2005 [online] http://www.ietf.org/rfc/rfc3927.txt [30/03/09]

[25] S. Cheshire, M. Krochmal. (2006) Multicast DNS. Internet draft [online]. Disponible en http://files.multicastdns.org/draft-cheshire-dnsext-multicastdns.txt [consulta 10/04/09]

[26] P. Leach, M. Mealling, and R. Salz. *A UUID URN Namespace* <draft-mealling-uuid-urn-05>, December 2004.

[27] V. Cardellini, E. Casalicchio, M. Colajanni, and P. Yu. *The State of the Art in Locally Distributed Web-server Systems. Technical Report* RC22209, IBM Research Division, October 2001.

[28] J. Mogul. *Network Behavior of a Busy Web Server and its Clients. Research Report 95/5*, Western Research Laboratory, October 1995.

[29] C. Perkins and E. Guttman. *DHCP Options for Service Location Protocol* IETF. RFC 2610. Junio 1999. Disponible en http://www.rfc-editor.org/rfc/rfc2610.txt [Consulta 27/04/09]

[30] M. Tuexen, Q. Xie, R. Stewart, M. Shore, J. Loughney, and A. Silverton. Architecture for Reliable Server Pooling <draft-ietf-rserpool-arch-09>, February 2005.

[31] R. Stewart, Q. Xie, M. Stillman, and M. Tuexen. Aggregate Server Access Protocol (ASAP) <draft-ietf-rserpool-asap-11>, February 2005.

[32] Q. Xie, R. Stewart, M. Stillman, M. Tuexen, and A. Silverton. Endpoint Handlespace Redundancy Protocol (ENRP) <draft-ietf-rserpool-enrp-11>, February 2005.

[33] M. Urueña and D. Larrabeiti. eXtensible Binary Encoding (XBE32) <draft-uruena-xbe32-00>, March 2004.

[34] M. Urueña and D. Larrabeiti. *eXtensible Service Location Protocol (XSLP) <draft-uruena-xslp-00>*, Marzo 2004.

[35] M. Urueña and D. Larrabeiti. *eXtensible Service Registration Protocol (XSRP) <draft-uruena-xsrp-00>*, Marzo 2004.

[36] M. Urueña and D. Larrabeiti. *eXtensible Service Subscription Protocol (XSSP)* <draft-uruena-xssp-00>, Marzo 2004.

[37] M. Urueña and D. Larrabeiti. *eXtensible Service Transfer Protocol (XSTP) <draft-uruena-xstp-00>*, March 2004.

[38] M. Urueña and D. Larrabeiti. *XSDF: Common Elements and Procedures* <draft-uruena-xsdf-common-00>, Marzo 2004.

[39] W. Zhao, H. Schulzrinne, and E. Guttman. *RFC 3528: Mesh-enhanced Service Location Protocol (mSLP).* Abril 2003.

[40] M. Urueña, D. Larrabeiti. *Contribución al diseño de arquitecturas distribuidas de nodos de red programables.* Tesis Doctoral. Mayo de 2005

[41] J.F. Martín. *Diseño e implementación java del eXtensible Service Location Protocol (XSLP).* Proyecto Fin de Carrera. 2008.

[42] M. Fernández. *Diseño e Implementación del API del eXtensible Binary Encoding (XBE32).* Proyecto Fin de Carrera. 2005.